

AD-A047 164

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 9/2

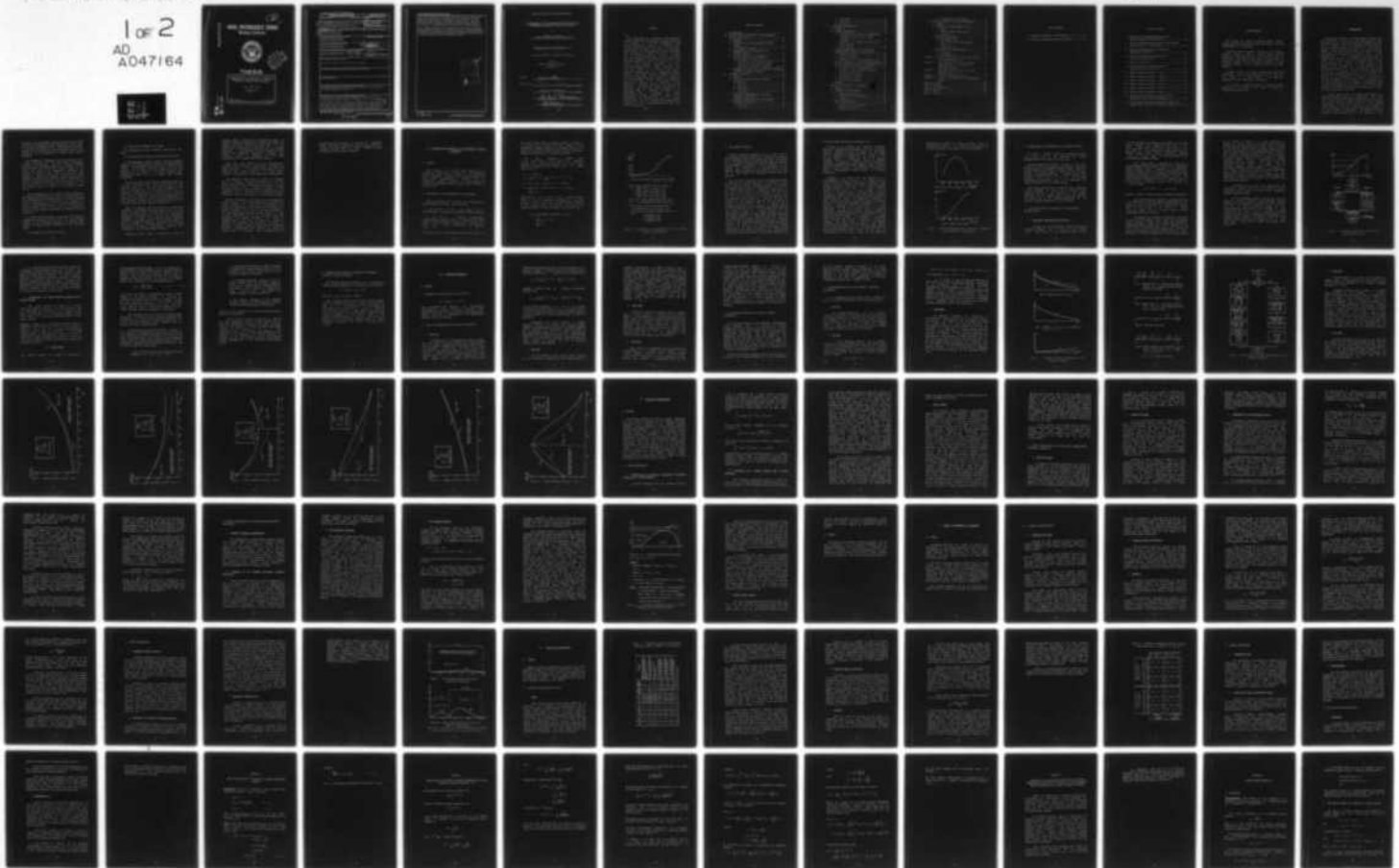
A COMPARISON OF TWO ALGORITHMS FOR THE SIMULATION OF NON-HOMOGE--ETC(U)

SEP 77 M L PATROW

UNCLASSIFIED

NL

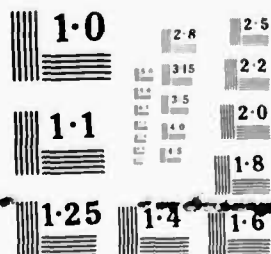
1 of 2
AD
A047164



1 OF 2

AD

A047164



AD A047164

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A COMPARISON OF TWO ALGORITHMS FOR THE SIMULATION OF
NON-HOMOGENEOUS POISSON PROCESSES WITH DEGREE-TWO
EXPONENTIAL POLYNOMIAL INTENSITY FUNCTION

by

Michael Lelon Patrow

September 1977

Thesis Advisor:

P.A.W. Lewis

Approved for public release; distribution unlimited.

AD No. _____
DDC FILE COPY

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. REPORT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Comparison of Two Algorithms for the Simulation of Non-Homogeneous Poisson Processes with Degree-Two Exponential Polynomial Intensity Function		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis September 1977
6. AUTHOR(s) Michael Lelon Patrow		7. PERFORMING ORG. REPORT NUMBER
8. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		9. CONTRACT OR GRANT NUMBER(s)
10. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		11. REPORT DATE September 1977
12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		13. NUMBER OF PAGES 122
		14. SECURITY CLASS. (of this report) Unclassified
15. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		16. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Non-homogeneous Poisson process; simulation; time-scale transformation; time series; Poisson decomposition.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Two algorithms for generating a non-homogeneous Poisson process with log-quadratic intensity function $\lambda(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2)$ are implemented into computer programs and compared for relative speed, core storage requirements and fidelity. By simulating several cases of non-homogeneous Poisson processes with log-quadratic intensity functions it is shown that the Poisson-decomposition and gap statistic algorithm, developed by Professor P.A.W. Lewis, Naval Postgraduate School, Monterey, California, and		

251450

JP

G.S. Shedler, IBM Research Laboratory, San Jose, California, substantially reduces computation time from that required by an algorithm that uses a time-scale transformation of a homogeneous Poisson process. The faster algorithm employs a rejection technique in conjunction with a method for simulating the non-homogeneous Poisson process with intensity function $\lambda(t) = \exp(\gamma_0 + \gamma_1 t)$ by generation of gap statistics. Although additional core storage is required by the Lewis and Shedler algorithm, the resulting gain in computing efficiency is so significant that it outweighs the memory consideration. The experience gained from implementing the algorithm has led to several possibilities which are suggested for improving the efficiency of the Poisson-decomposition and gap statistic algorithm.

ACCESSION FOR	
NTIS	Write Section <input checked="" type="checkbox"/>
DDC	Buy Section <input type="checkbox"/>
MAINTAINING TO	
J S SECTION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
SPECIAL	
A	

Approved for public release; distribution unlimited

A COMPARISON OF TWO ALGORITHMS FOR THE SIMULATION OF
NON-HOMOGENEOUS POISSON PROCESSES WITH DEGREE-TWO
EXPONENTIAL POLYNOMIAL INTENSITY FUNCTION

by

Michael L. Patrow
Captain, United States Marine Corps
B. S. United States Military Academy, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the
NAVAL POSTGRADUATE SCHOOL
September 1977

Author:

Michael L. Patrow

Approved by:

Paul A. W. Lewis
Thesis Advisor

Donald R. Bouchant
Second Reader

Michael J. Foreman
Chairman, Department of Operations Research

A. Schrad
Dean of Information and Policy Sciences

1

ABSTRACT

Two algorithms for generating a non-homogeneous Poisson process with log-quadratic intensity function $\lambda(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2)$ are implemented into computer programs and compared for relative speed, core storage requirements and fidelity. By simulating several cases of non-homogeneous Poisson processes with log-quadratic intensity functions it is shown that the Poisson-decomposition and gap statistic algorithm, developed by Professor P.A.W. Lewis, Naval Postgraduate School, Monterey, California, and G.S. Shedler, IBM Research Laboratory, San Jose, California, substantially reduces computation time from that required by an algorithm that uses a time-scale transformation of a homogeneous Poisson process. The faster algorithm employs a rejection technique in conjunction with a method for simulating the non-homogeneous Poisson process with intensity function $\lambda(t) = \exp(\gamma_0 + \gamma_1 t)$ by generation of gap statistics. Although additional core storage is required by the Lewis and Shedler algorithm, the resulting gain in computing efficiency is so significant that it outweighs the memory consideration. The experience gained from implementing the algorithm has led to several possibilities which are suggested for improving the efficiency of the Poisson-decomposition and gap statistic algorithm.

TABLE OF CONTENTS

I.	INTRODUCTION.....	11
II.	DEFINITION AND PROPERTIES OF NON-HOMOGENEOUS POISSON PROCESSES.....	16
	A. GENERAL.....	16
	B. DEFINITION OF A NON-HOMOGENEOUS POISSON PROCESS.....	16
	C. THE INTENSITY FUNCTION.....	19
	D. DECOMPOSITION AND SUPERPOSITION OF POISSON PROCESSES.....	22
	E. TWO BASIC METHODS OF GENERATING A NON-HOMOGENEOUS POISSON PROCESS.....	22
	1. Time-Scale Transformation Algorithm.....	22
	2. Conditioning and Order Statistics Algorithm for a Poisson Process.....	26
	F. RATIONALE FOR SELECTION OF DEGREE-TWO EXPONENTIAL POLYNOMIAL INTENSITY FUNCTION.....	29
III.	COMPETING ALGORITHMS.....	30
	A. GENERAL.....	30
	B. TIME-SCALE TRANSFORMATION ALGORITHM (ALGORITHM A).....	30
	1. Step One.....	30
	2. Step Two.....	31
	3. Step Three.....	32
	4. Step Four.....	32
	C. TIME-SCALE TRANSFORMATION ALGORITHM, ALTERNATE (ALGORITHM A').....	33
	D. POISSON-DECOMPOSITION AND GAP STATISTIC ALGORITHM (ALGORITHM B).....	34
	1. Step One.....	34
	2. Step Two.....	34

3.	Step Three.....	35
4.	Step Four.....	39
5.	Step Five.....	39
IV.	ALGORITHM IMPLEMENTATION.....	46
A.	GENERAL.....	46
B.	COMMON REQUIREMENTS.....	46
1.	Integration of a Degree-Two Exponential Polynomial Function over a Fixed Interval.....	46
2.	Generation of a Poisson Variate with a Given Parameter.....	47
3.	Event Storage.....	49
C.	SPECIAL REQUIREMENTS OF THE TIME-SCALE TRANSFORMATION ALGORITHM (ALGORITHM A).....	50
1.	Uniform Variates.....	50
2.	Sorting of Events.....	51
3.	Computation of the Transformed Values....	52
D.	SPECIAL REQUIREMENTS OF THE POISSON-DECOMPOSITION AND GAP STATISTIC ALGORITHM (ALGORITHM B)....	56
1.	Intensity Function Categorization.....	56
2.	Selection of the Imbedded Log Linear Intensity Function(s).....	56
3.	Gap Statistic Algorithm.....	57
4.	The Rejection Routine.....	58
5.	Merging Event Streams.....	61
E.	SUMMARY.....	62
V.	METHOD OF COMPARISON OF ALGORITHMS.....	63
A.	GENERAL.....	63
B.	MEASURES OF EFFECTIVENESS.....	64
1.	Computational Speed.....	64
2.	Computer Memory Requirements.....	65
3.	Fidelity.....	65
C.	OTHER CONSIDERATIONS.....	69
1.	Intensity Function Category.....	69
2.	Evaluation of c^* Value in Rejection Routine.....	69

3. Designated Tolerance Level.....	70
VI. RESULTS, CONCLUSIONS AND RECOMMENDATIONS.....	73
A. GENERAL.....	73
B. MEASURES OF EFFECTIVENESS RESULTS.....	73
1. Speed.....	73
2. Computer Memory Requirements.....	76
3. Fidelity.....	76
C. GENERAL OBSERVATIONS.....	80
1. Programming Ease.....	80
2. Exact Method Versus Approximate Method.....	80
3. Initialization.....	81
D. CONCLUSION AND RECOMMENDATIONS.....	81
1. Conclusion.....	81
2. Recommendations.....	82
Appendix A: PROOF OF VALIDITY OF SCALING THE INVERSE DISTRIBUTION FUNCTION.....	94
Appendix B: ERROR FUNCTION AND DAWSON'S INTEGRAL REPRESENTATION OF THE INTEGRATED INTENSITY FUNCTION.....	96
Appendix C: COMPARISON OF ALGORITHMS, LOG-LINEAR INTENSITY FUNCTION.....	92
Appendix D: POISSON VARIATE GENERATION.....	94
Appendix E: GRAPHICAL PRESENTATION OF NEWTON-RAPHSON METHOD.....	100
PROGRAM LISTINGS.....	105
LIST OF REFERENCES.....	118
INITIAL DISTRIBUTION LIST.....	120
LIST OF TABLES.....	8
LIST OF FIGURES.....	9

LIST OF TABLES

I	Computation Times for Event Streams.....	74
II	Results of Hypotheses Tests for Fidelity.....	79

LIST OF FIGURES

1. Definition of a Non-Homogeneous Poisson Process (Graphical Representation).....	18
2. Rate and Integrated Rate Functions for Telephone Calls Arriving at a Switchboard.....	21
3. Graphical Representation of Time-Scale Transformation Method.....	25
4. Diagram of Poisson-Decomposition and Gap Statistic Algorithm.....	36
5. Diagram (Continued).....	37
6. Flow Diagram of Poisson-Decomposition and Gap Statistic Algorithm.....	38
7. Sample Intensity Function - Case I.....	40
8. Sample Intensity Function - Case II.....	41
9. Sample Intensity Function - Case III.....	42
10. Sample Intensity Function - Case IV.....	43
11. Sample Intensity Function - Case V.....	44
12. Sample Intensity Function - Case VI.....	45
13. The Rejection-Acceptance Method of Variate Generation from an Arbitrary Density.....	60
14. Illustration of Rejection-Acceptance Regions for Sample Case V and Case VI Intensity Functions.....	72

ACKNOWLEDGEMENT

This thesis was hardly an individual effort. Several persons graciously contributed their time, skills, ideas and encouragement to me throughout.

Professor Richard W. Hamming advised me on numerical analysis, Lieutenant Donald R. Bouchoux, USN, on format, content and sundry other matters, and Mr. Edward N. Ward on programming. Technical typing support was cheerfully and expertly provided by Miss Rosemarie Stampfel, a truly delightful person. To all of these individuals I am indeed grateful.

Professor Peter A. W. Lewis is deserving of particular recognition. He is not only responsible for the topic but also served as my thesis advisor. I am especially appreciative of his assistance and interest.

And finally, my thanks to Renee, who helped me perhaps most of all, as she does in all things, by being the wonderful wife that she is.

I. INTRODUCTION

Many familiar physical and operational processes are well described, in whole or part, by examining their "event streams" over time. Some such well-known processes are: the flow of traffic through an intersection; the arrival of persons at some service facility such as a bank teller's window, a service station fuel pump or a grocery store check out counter; and, the arrival of telephone calls or radio transmissions at some switchboard or other type of communications terminal. Analogous processes abound both in nature and in the course of our everyday lives. By the proper definition of an "event" in these situations, the process being observed will be characterized by the probabilistic nature of the flow, over time, of the events of which it is composed. In the above three examples the events could be defined respectively as the arrival of a vehicle at the intersection, the arrival of a customer at the service facility, and the arrival of the telephone call or radio transmission at the terminal. The process may then be analyzed by examining the interaction of various event streams with different intersection configurations, service policies and terminal capacities.

A common method used to perform such analyses is to consider the event streams to be homogeneous. This could mean that the expected number of events to occur in any two or more time intervals of equal length is the same (simple homogeneity), or that the distribution of the number of events occurring in any two or more equal time intervals is the same (complete homogeneity). The homogeneous Poisson process is often used as a tool in the analysis of such

activities. For very simple systems involving event streams the use of the homogeneous Poisson process as a model leads to tractable analytical results. Most systems of interest, however, are not amenable to purely analytical methods, and simulation of the processes by digital means becomes necessary.

The assumption of homogeneity in event streams is often a very restrictive one. The "rush hour" phenomenon, well-known and abundantly cursed by motorists, provides cogent evidence that the modeling of event streams is not always well served by the homogeneity assumption. The intensity of event streams varies over time for many physical or operational processes. In these cases, purely analytical methods must be abandoned almost immediately in favor of simulation techniques. (The intensity of the event stream is defined to be the derivative with respect to t of the expected number of events in an interval of length $(0, t]$.)

The non-homogeneous Poisson process is often employed in the analysis of processes that exhibit gross departures from the homogeneous event stream criterion. If these processes are to be simulated, it is necessary to first describe the nature of the inhomogeneity (i.e. how does the intensity of the event stream vary over time?) and then to artificially generate event streams that behave in accordance with the description.

Of course there are infinite variations in the types of inhomogeneities that can occur or that can be construed. However it is intuitively appealing to consider event streams that display varying intensities of the following types:

- i) increasing continuously over time;

- ii) decreasing continuously over time;
- iii) increasing and then decreasing continuously over time;
- iv) decreasing then increasing continuously over time.

A non-homogeneous Poisson process that has quadratic properties can be manipulated to produce the above-mentioned effects. The effective simulation of such a process on the computer is the subject of this thesis and has been motivated by the work of P. A. W. Lewis, Professor, Naval Postgraduate School, and G. S. Shedler, IBM Research Laboratory.

There are of course other types of inhomogeneities, such as cyclic variations (time of day effect), but we do not consider them here. They have been discussed by COX [Ref. 2] and LEWIS [Ref. 9]. In particular LEWIS [Ref. 9] describes a process consisting of arrivals at an intensive care unit in a hospital. It is shown empirically that, in addition to the time of day cycle, long term fluctuations in the intensity function can be adequately described by an intensity function whose logarithm is quadratic.

LEWIS and SHEDLER [Ref. 11] proposed a new method for generating a non-homogeneous Poisson process with an event stream intensity (rate) function that is of degree-two exponential polynomial form. (The use of exponential polynomials is natural in this context since an intensity function is a positive function.) The new method appears to have the virtue of increased efficiency over the more conventional time-scale transformation technique when implemented on a high speed digital computer.

Efficiency in this context is measured in terms of

computer memory requirements and computational speed. The problem of efficiency comparison is recognized by LEWIS and SHEDLER in the final pages of their paper [Ref. 11, p. 15]: "There remains the question of efficiency (of the proposed algorithm) . . . for generation of a non-homogeneous Poisson process with degree-two exponential polynomial rate function, relative to generation via time scale transformation of a homogeneous Poisson process."

After some brief discussion of the requirements of implementing the time-scale transformation algorithm the report concludes . . . "We therefore would expect the exact method of (the proposed algorithm) to be much faster, although at the expense of some complexity of programming."

The objective of this author has been: to implement both the algorithm of LEWIS and SHEDLER [Ref. 11] and the conventional time-scale transformation algorithm on the IBM 360/67 Computer System in FORTRAN IV language; to define reasonable measures of effectiveness for comparing the two algorithms; and to determine which algorithm is the more efficient in terms of the measures of effectiveness defined.

Section II discusses the definition of a non-homogeneous Poisson process. It also states some special properties of Poisson processes that are used in the development of the algorithms investigated in this thesis, and concludes with a general discussion of two basic methods of generating a non-homogeneous Poisson process. Section III gives a step by step description of the two algorithms that were implemented into computer programs. The method of implementation is the topic of Section IV. Methods of comparing the algorithms are presented in Section V. Section VI presents conclusions drawn from the comparison and makes a recommendation for improving the LEWIS and SHEDLER [Ref. 11] algorithm. Other recommendations for

further study are also listed in Section VI. Appendixes provide additional details that would have been awkward to include in the main body of the thesis. Computer program listings are provided after Appendix E.

II. DEFINITION AND PROPERTIES OF NON-HOMOGENEOUS POISSON PROCESSES

A. GENERAL

This section will present basic definitions and explanations concerning the concept of non-homogeneous Poisson processes. References cited may be consulted if a more in-depth understanding is desired. Only the fundamental concepts necessary for understanding the specific non-homogeneous Poisson process under consideration are presented.

B. DEFINITION OF A NON-HOMOGENEOUS POISSON PROCESS

LEWIS and SHEDLER [Ref. 11] define the non-homogeneous Poisson process on the real line as follows:

1. The number of events in any finite set of non-overlapping intervals are independent random variables.

2. Let $\Lambda(t)$ be a monotone non-decreasing right-continuous function which is bounded on any finite interval. Then the number of events in any interval, e.g. $(0, t_0)$, has a Poisson distribution with parameter $\Lambda(t_0) - \Lambda(0)$.

The function $\Lambda(t) - \Lambda(0)$ is called, among other things, the

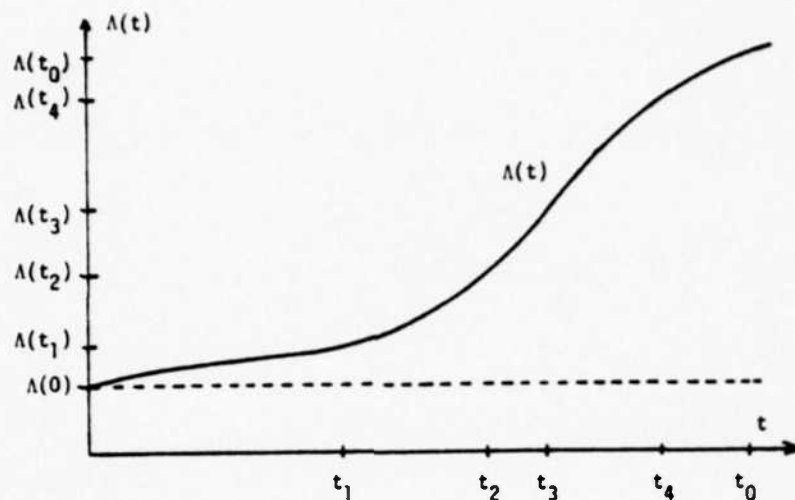
mean value function since the expected number of events in an interval $(0, t]$ is just $\Lambda(t) - \Lambda(0)$. Property 1 is the independent increment property of the Poisson process; it is basic to the idea of a Poisson process. Figure 1 provides a graphical representation of the definition above.

The above definition insures that $\{ \Lambda(t) - \Lambda(0) \} / \{ \Lambda(t_0) - \Lambda(0) \}$ meets the following criteria for an arbitrary function $F(t)$, to be a valid distribution function on the interval $(0, t_0)$, c.f. LARSON [Ref. 7, ch. 3];

- i) $0 \leq F(t) \leq 1$
- ii) $\lim_{t \rightarrow 0} F(t) = 0; \lim_{t \rightarrow t_0} F(t) = 1, \quad 0 < t < t_0$
- iii) $F(a) \leq F(b)$ for all $a \leq b$ in $(0, t_0)$
- iv) $\lim_{h \rightarrow 0} F(b+h) = F(b)$ for all b in $(0, t_0)$,
where $h > 0$.

Letting $F(t) = \{ \Lambda(t) - \Lambda(0) \} / \{ \Lambda(t_0) - \Lambda(0) \}$ it follows that if $\Lambda(t)$ is absolutely continuous in $(0, t_0]$ then $dF(t)/dt = \lambda(t) / \{ \Lambda(t_0) - \Lambda(0) \}$ is a valid density function on the interval $(0, t_0]$. The function $\lambda(t)$ is called the intensity function (or rate function) of the process and

$$\begin{aligned} \lambda(t) &= \frac{d}{dt} E\{\text{number of events in } (0, t_0]\} \\ &= \frac{d}{dt} \{ \Lambda(t) - \Lambda(0) \} \\ &= \frac{d}{dt} \Lambda(t) . \end{aligned}$$



Situation: Events occur randomly in time (i.e. along t-axis)

- 1) If X = number of events in fixed interval $(t_1, t_2]$
 Y = number of events in fixed interval $(t_2, t_3]$
 Z = number of events in fixed interval $(t_3, t_4]$
 S = number of events in fixed interval $(0, t_0]$;

Then X , Y and Z must be independent random variables (note:
 S and Z are not independent because of overlap in the intervals).

- 2) Given the monotonic increasing right continuous function $\Lambda(t)$, the number of events in any fixed interval $(t_i, t_j]$ must have the Poisson distribution with parameter $\Lambda(t_j) - \Lambda(t_i)$.

Thus

$$\begin{aligned} X &\sim \text{Poisson}(\Lambda(t_2) - \Lambda(t_1)) \\ Y &\sim \text{Poisson}(\Lambda(t_3) - \Lambda(t_2)) \\ Z &\sim \text{Poisson}(\Lambda(t_4) - \Lambda(t_3)) \\ S &\sim \text{Poisson}(\Lambda(t_0) - \Lambda(0)) \end{aligned}$$

Figure 1 - DEFINITION OF A NON-HOMOGENEOUS POISSON PROCESS
 (GRAPHICAL REPRESENTATION)

C. THE INTENSITY FUNCTION

The most intuitively appealing way to think about a non-homogeneous Poisson process is to consider the variation in the intensity of the event stream over time. The concept of an intensity function whose integral meets the criteria of the definition in paragraph B above is essential to the modeling and simulation of a non-homogeneous Poisson process. When one starts with the existence of $\lambda(t)$, the $\Lambda(t)$ is often called the integrated intensity (or rate) function.

The intensity function reveals the instantaneous rate of arrivals (in the event stream) as a function of time. (This function must be a positive function.) For example, if telephone calls arrive at a switchboard at the rate of 5 per hour at 0900 (9:00 a.m.), increase to a peak rate of 20 per hour at 1300 (1:00 p.m.), then decrease to a rate of 5 per hour at 1700 (5:00 p.m.), the intensity function could look something like Figure 2a. Then, by plotting the integral of the intensity function over the interval of interest (i.e. from 0900 to 1700) it is obvious that a monotone-increasing, right-continuous and bounded function is obtained (see Figure 2b). If the assumption is made that the arrival stream is a Poisson process, i.e. has independent increments, then the number of calls received in any chosen interval (e.g. 0900-1000, 1230-1315, 1107-1632) is distributed as a Poisson random variable with parameter equal to the difference between the integral evaluated at the right end point of the interval and the integral evaluated at the left end point of the interval. These values may be read directly from Figure 2b. Specifically, the number of calls received in an eight-hour working day is

a Poisson random variable with parameter 120.

Although it is unlikely that telephone calls would arrive at a switchboard in accordance with the convenient parabolic intensity function of Figure 2a, the example serves to illustrate two important points. First, it would not be realistic to assume that the arrival rate of telephone calls at a switchboard would be constant throughout a working day. Some function that describes an initially increasing and finally decreasing rate of arrivals seems more akin to reality. The importance of being able to model a non-homogeneous Poisson process is therefore established.

Secondly, it is obvious that the definition of a non-homogeneous Poisson process is not always used as a starting point for modeling operational processes. Event streams are usually thought of in terms of their underlying intensity functions. The idea of an intensity function applies to any model for an event stream, and is not specific to a Poisson process. The further step of modeling the physical process as a non-homogeneous Poisson process by assuming that the process has independent increments is taken either on the basis of empirical evidence or physical reasoning. Testing for independent increments in a point process is discussed in COX and LEWIS [Ref. 3, ch. 6] and LEWIS [Ref. 9]. The main physical reason for assuming independent increments, and therefore a Poisson process, is that the operational process is the superposition of many individual event streams. For instance, in a computer center equipped with several interactive time-sharing terminals, the event stream of users at each specific terminal might be assumed to be an arbitrary point process with a certain intensity function. The event stream seen by the central processing unit is then the sum total (or superposition) of the event streams of the individual

terminals and, if there are enough terminals, should be approximately Poisson. This property of superposition of intensity functions is discussed in the next paragraph.

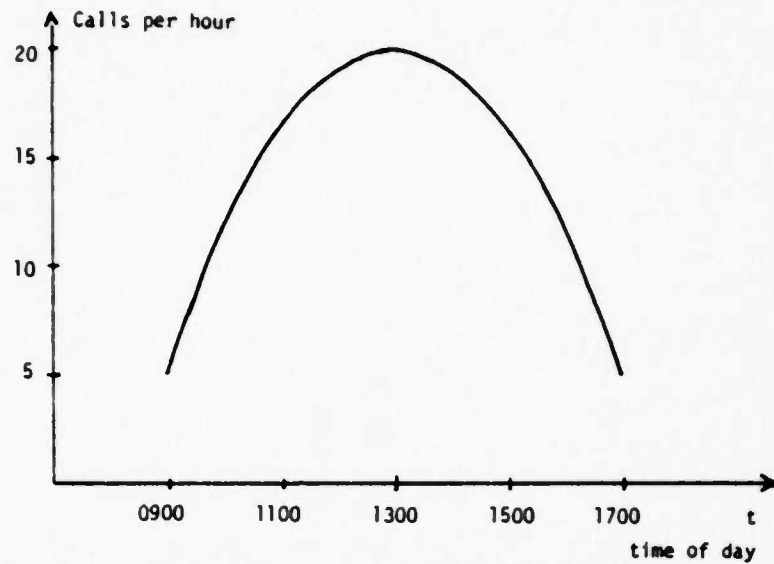


Figure 2a - Rate of Arrival of Telephone Calls at a Switchboard

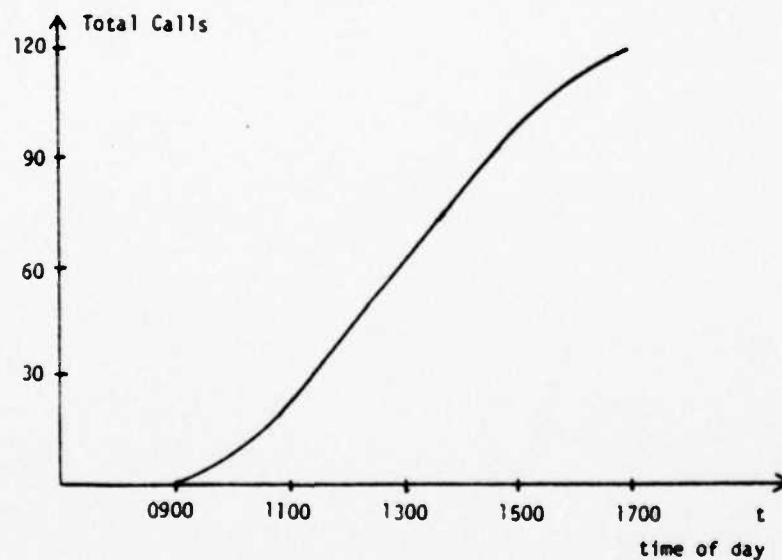


Figure 2b - Average Total Calls Received

Figure 2 - RATE AND INTEGRATED RATE FUNCTIONS FOR TELEPHONE CALLS ARRIVING AT A SWITCHBOARD

D. DECOMPOSITION AND SUPERPOSITION OF POISSON PROCESSES.

To obtain a Poisson process with intensity function $\lambda(t) = \lambda_1(t) + \lambda_2(t)$, we may superpose two Poisson processes, each of intensity $\lambda_1(t)$ and $\lambda_2(t)$.

We may decompose the intensity function of any event stream into two or more component event streams. However if we then superpose the component streams, we will recover the original type process only if we started with a Poisson process. For example begin with a renewal process with intensity $\nu(t) =$ and let $\nu = \nu_1 + \nu_2$. If two renewal processes with intensities ν_1 and ν_2 are superposed, the resulting process is not a renewal process.

This unique property may be exploited when simulating Poisson processes. It permits the partitioning of the intensity function to take advantage of any special properties of its component parts. This is the basis for the method used in the Poisson-Decomposition and Gap Statistic Algorithm discussed in later sections.

E. TWO BASIC METHODS OF GENERATING A NON-HOMOGENEOUS POISSON PROCESS

1. Time-Scale Transformation Algorithm

Consider the non-homogeneous Poisson process with intensity function $\lambda(t) > 0$, $0 < t < t_0$, on the interval $(0, t_0]$. The integral of the intensity function is then

$\Lambda(t)$ and the number of events in the interval is a Poisson random variable with parameter $\Lambda(t_0) - \Lambda(0) = \mu_0$ (or $\Lambda(t_0) = \mu_0$ since $\Lambda(0) = \int_0^0 \lambda(t) dt = 0$). Now if T_1^*, \dots, T_n^* are events in a unit homogeneous Poisson process (i.e. a Poisson process with constant intensity function $\lambda'(t) = \lambda' = 1$) then $\Lambda^{-1}(T_1^*), \dots, \Lambda^{-1}(T_n^*)$ are events in the non-homogeneous Poisson process.

This result gives a procedure for simulating a non-homogeneous Poisson process, starting with a homogeneous Poisson process, which is analogous to the probability integral transform method of producing random samples from a continuous distribution with distribution function $F(x)$ when starting with uniformly distributed random variables. The latter method is essentially that if the inverse of $F(x)$ can be found, then by generating uniform $(0,1)$ variates u_1, \dots, u_n the values

$$x_1 = F^{-1}(u_1), \dots, x_n = F^{-1}(u_n)$$

comprise a sample of variates from the desired distribution.

The right-continuous monotone increasing function $\Lambda(t)$ describing the non-homogeneous Poisson process on the interval $(0, t_0]$ can be thought of as a distribution function which has been "scaled" by the factor μ_0 . (Since $\Lambda(t_0) = \mu_0$, then $\Lambda(t_0)/\mu_0 = 1$, thus $\Lambda(t)/\mu_0$ is a valid distribution function on $(0, t_0]$.)

To implement the time-scale transformation procedure one can use the following basic result for homogeneous Poisson processes: given that n events have occurred in a homogeneous Poisson process over a fixed interval $(0, t_0]$, those events are uniformly distributed on the interval. A proof of this property is given in PARZEN [Ref. 14, p. 140]. Therefore, if events are generated as a unit Poisson process

on an interval of length μ_0 by first obtaining a Poisson (μ_0) random variate n , and then letting the order statistics from a random sample of uniform $(0, \mu_0)$ variates be the times to events in the unit Poisson process, and the times to these events are then transformed by the inverse of the integrated intensity function, i.e. $\Lambda^{-1}(\cdot)$, the effect is the same as that obtained from the probability integral transform method. Figure 3 illustrates this method graphically and also provides a flow chart. Note the difference however between this procedure and the probability integral transform procedure. In generating a unit Poisson process by this method we need a sample whose size is random (and could be zero) i.e. Poisson (μ_0). The probability integral transform method simply transforms a fixed number of uniform $(0,1)$ variates into variates from some other distribution.

In Appendix A it is proved that scaling both the distribution function $F(x)$ and the interval $(0,1)$ by the same factor does not affect the validity of the probability integral transform method.

The unit homogeneous Poisson process may also be generated by adding a sequence of unit exponential variates (variates from an exponentially distributed random variable with mean = 1) until the sequence of partial sums of the random variables first exceeds μ_0 (see Appendix D). This accomplishes two things. First it provides an ordered sequence of events from a homogeneous Poisson process on the interval $(0, \mu_0]$. Second, it determines a realization of the Poisson random variable N_{t_0} with parameter $\Lambda(t_0) = \mu_0$. Note that given n , the times to events are uniform order statistics.

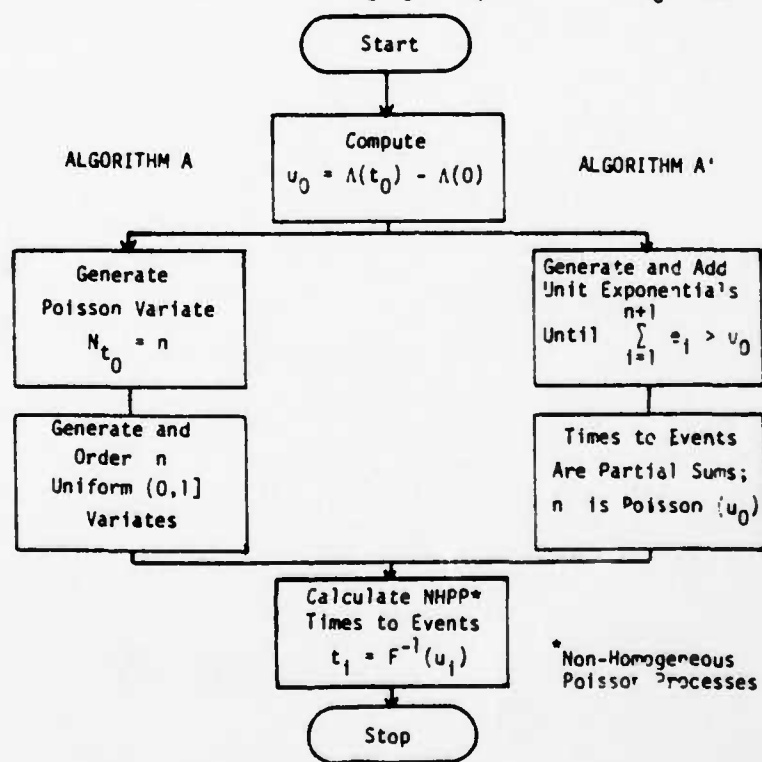
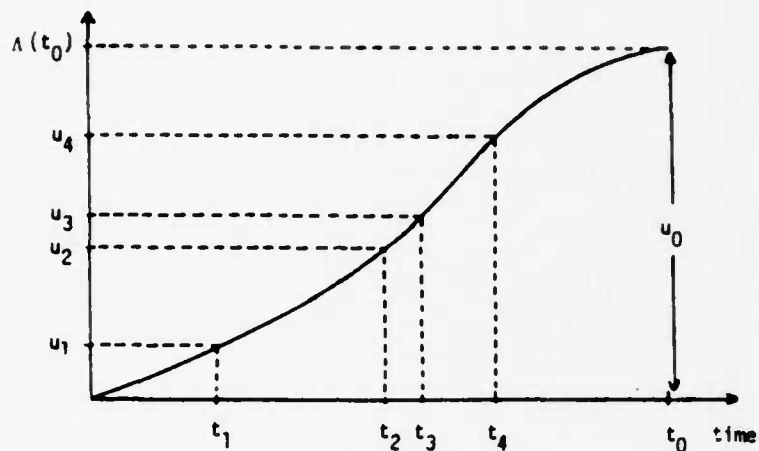


Figure 3 - GRAPHICAL REPRESENTATION OF TIME-SCALE TRANSFORMATION METHOD

The difference between these methods for generating a homogeneous Poisson process should also be noted. The first method requires a Poisson variate and ordering of that number of uniform random variables. The second requires generation of independent exponential variates. The second method is probably most basic in that it requires only exponentially distributed random variables, and these can be obtained from uniform variates by the inverse probability integral transform, which is just a logarithm. The method is, however, not always the most efficient.

2. Conditioning and Order Statistics Algorithm for a Poisson Process

This method requires the result of a theorem sketched by LEWIS and COX [Ref. 3, ch. 2] and restated here for convenience and continuity; it is an extension of the result on conditioning in a homogeneous Poisson process which results in a conditional uniform distribution of the times to events.

Theorem 1: Assume that a non-homogeneous Poisson process is observed for a fixed time $(0, t_0]$, so that the number of events in $(0, t_0]$, N_{t_0} , has a Poisson distribution with parameter $\Lambda(t_0) - \Lambda(0) = \mu_0$. Then if T_1, \dots, T_n denote times-to-events for the process in $(0, t_0]$, and if $N_{t_0} = n$, conditional on having observed $n(>0)$ events in $(0, t_0]$, the T_i 's are distributed as order statistics from a sample with distribution function

$$F(t) = \frac{\Lambda(t) - \Lambda(0)}{\Lambda(t_0) - \Lambda(0)} \quad .$$

This theorem reduces the problem of simulating a

non-homogeneous Poisson process to that of generating a Poisson number of order statistics from a fixed distribution function. That is, given an intensity function over an interval $(a,b]$, whose definite integral $\Lambda(t) = \int_a^t \lambda(s)ds$ is bounded and right-continuous on the interval, an ordered sample, from a population with distribution function

$$F(t) = \frac{\Lambda(t) - \Lambda(a)}{\Lambda(b) - \Lambda(a)} \quad a < t \leq b \quad (1)$$

will yield the desired non-homogeneous Poisson process defined by the intensity function $\lambda(t)$ on $(a,b]$. For simplicity the interval will hereafter be assumed to have its left end point at zero ($a = 0$) and its right end point at some arbitrary, but fixed point t_0 , ($b = t_0$). Using this $(0,t_0]$ interval results in no loss of generality, and (1) becomes identical with the expression in the theorem.

Many methods exist for obtaining the necessary order statistics. The inverse integral transform explained above, decomposition of the density function (see LEWIS Ref. 8), or the rejection-acceptance method discussed later in Section IV-D, are all possibilities.

For the family of intensity functions addressed in this thesis the non-homogeneous Poisson process is obtained by a combination of Poisson decomposition, an algorithm of LEWIS and SEEDLER [Ref. 13] for obtaining a non-homogeneous Poisson process with log-linear intensity function, and the conditioning and order statistics theorem given above. The procedure involves four basic steps.

1. The intensity function is decomposed into two components; i.e. $\lambda(t) = \underline{\lambda}(t) + \lambda^*(t)$.

2. A gap statistics algorithm is used to generate a non-homogeneous Poisson process from one of the components, $\lambda(t)$, which is chosen such that it has a special structure (log-linear).

3. A rejection-acceptance routine is used to produce a sample from the remaining component, $\lambda^*(t)$, i.e. a Poisson number of ordered variates are generated. This algorithm is described in detail in Section III-C. This sample, when ordered, becomes a Poisson process also.

4. The Poisson processes of the component intensity functions are then superposed to produce the desired non-homogeneous Poisson process.

Figures 4, 5 and 6 (Section III) illustrate the four general steps of this procedure.

Note: Theorem 1 provides a second way to generate the unit homogeneous Poisson process required by the time-scale transformation algorithm previously described. First we may generate a variate from the Poisson random variable N_{t_0} with parameter μ_0 , say $N_{t_0} = n$. Then conditional upon $N_{t_0} = n$, n uniform order statistics can be generated on the interval. For reasons explained in Appendix D, this second method was used in the computer program implementation of the time-scale transformation method.

F. RATIONALE FOR SELECTION OF DEGREE-TWO EXPONENTIAL POLYNOMIAL INTENSITY FUNCTION

The intensity function identifying the non-homogeneous Poisson process investigated in this paper is of the form,

$$\lambda(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2)$$

where α_0 , α_1 and α_2 are real constants.

This intensity function was selected for three reasons. First, an intensity function must always be positive (or zero) if it is to be meaningful. The above function is non-negative for all values of α_0 , α_1 and α_2 . Secondly, this intensity function, by proper choice of constants, can be used to represent the four different types of event streams mentioned previously in Section I. And finally, the selection of this intensity function leads to simple statistical procedures; (for details, see LEWIS Ref. 9, p.30-34).

III. COMPETING ALGORITHMS

A. GENERAL

Assuming an intensity function of the form

$$\lambda(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2)$$

three algorithms for generating the corresponding non-homogeneous Poisson process are discussed. These algorithms are based on the two general methods presented in Section II-E, and the decomposition and superposition property of Poisson processes.

B. TIME-SCALE TRANSFORMATION ALGORITHM (ALGORITHM A)

1. Step One

By definition of a non-homogeneous Poisson process, the total number of events observed over a fixed interval $(0, t_0)$ is itself a Poisson distributed random variable, N_{t_0} , with parameter $\mu_0 = \int_0^{t_0} \lambda(t) dt$. The first step of the algorithm is to determine the value of the parameter μ_0 . Although an explicit, closed-form expression for the above integral cannot be found, a series representation does exist. Except for a constant factor, this series

representation assumes the form of the error function or of Dawson's integral. A negative value for the coefficient of the second degree term, α , yields the error function form:

$$\mu_0 = K \left(\frac{2}{\sqrt{\pi}} \int_0^{t_2} e^{-u^2} du - \frac{2}{\sqrt{\pi}} \int_0^{t_1} e^{-u^2} du \right)$$

whereas a positive value for results in the Dawson's integral form:

$$\mu_0 = Kt_2^2 \left(t_2^{-2} \int_0^{t_2} e^{u^2} du \right) - Kt_1^2 \left(t_1^{-2} \int_0^{t_1} e^{u^2} du \right)$$

In the above expressions K , t_1 , and t_2 are uniquely determined by the coefficients α_0 , α_1 , α_2 and the end points of the interval over which the intensity function applies. A detailed derivation of above relationships is given in Appendix C.

Evaluation of the error function is a FORTRAN supplied procedure and requires only that the proper arguments be calculated and provided to the FORTRAN FUNCTION ERF or LERF [Ref. 15]. Evaluation of Dawson's integral is best accomplished through use of the IMSL (International Mathematical and Statistical Libraries, Inc.) FUNCTION MMDAW [Ref. 6]. The accuracy of the function values calculated by these routines is limited only by the precision characteristics of the computer.

2. Step Two

Once the parameter of the Poisson random variable N_{t_0} is determined, a realization on that random variable is

required. (The approach is somewhat backwards since it first determines how many events occurred over the interval. It then distributes that fixed number of events over the interval in accordance with the non-homogeneous Poisson process described by the intensity function. The importance of Theorem 1 now becomes evident since it assures the validity of such a procedure.) Generation of Poisson variates, especially those with large parameter values, is a complex procedure in itself if efficiency in terms of computer time and memory requirements is desired. This problem is discussed later in Section IV-B. For the present, assume that the requisite variate has been produced, i.e. $N_{t_0} = n$.

3. Step Three

Given that n events have occurred over the interval $(0, t_0]$ we then distribute n events along an interval of length μ_0 in accordance with a homogeneous Poisson process. Since events in a homogeneous Poisson process are uniformly distributed over an interval (given that n events have occurred), this step merely requires that n uniform $(0,1)$ variates be generated, ordered from lowest to highest, and then each multiplied by the factor μ_0 . The values in this n -element vector, $(u'_1, u'_2, \dots, u'_n)$, correspond to the points plotted on the vertical axis in Figure 3.

4. Step Four

Each event in the homogeneous Poisson process must be transformed by the inverse of the integral of the intensity function. Letting $\int_0^t \lambda(s) ds = \Lambda(t)$, the inverse $\Lambda^{-1}(\cdot)$ applied to each event in the homogeneous Poisson process, will produce a corresponding event in the

non-homogeneous Poisson process, i.e. $\Lambda^{-1}(u_1) = t_1$, $\Lambda^{-1}(u_2) = t_2$, etc. The difficulty is that since the integral of this specific intensity function cannot usually be explicitly expressed, the form of its inverse usually eludes any convenient computational formula expression. The unique position on $(0, t_0]$ the inverse determines for each input value can be found to any degree of accuracy desired, by iterative, numerical methods. The Newton-Raphson method is easily employed and very efficient in the present scenario. Its implementation is explained in Section IV-C. Since the function $\Lambda(t)$ is strictly monotone increasing, the inverse function $\Lambda^{-1}(u)$ applied to an ordered sequence of input values results in an ordered sequence of output values. Therefore, t_1, t_2, \dots, t_n are the times of events in the non-homogeneous Poisson process and the algorithm is complete.

C. TIME-SCALE TRANSFORMATION ALGORITHM, ALTERNATE (ALGORITHM A')

An alternative approach to the time-scale transformation method described above is to generate the required homogeneous Poisson process by using the fact that in this process the random times between events are independently exponentially distributed. Thus one generates unit exponential variates until their sum exceeds μ_0 . The partial sums give the times to events and the number of partial sums less than or equal to μ_0 is a Poisson (μ_0) variate. Note that the Poisson variate comes out as a by-product in this procedure rather than as a pre-product as in Step Two of Algorithm A above.

Although this method combines Step Two and Step Three of Algorithm A into a single procedure, it is not necessarily

the best method. Because it requires the use of the additive method of Poisson variate generation, it becomes inefficient for Poisson processes with many events (see Appendix D). For this reason Algorithm A instead of Algorithm A' was used when implementing the time-scale transformation method into a computer program.

D. POISSON-DECOMPOSITION AND GAP STATISTIC ALGORITHM (ALGORITHM B)

It is recommended that the reader refer to Figures 4, 5 and 6 for a better understanding of the following steps.)

1. Step One

The Poisson-decomposition and gap statistic algorithm begins with an examination of the coefficients of the intensity function. By doing so, the intensity function is categorized into one of six possible configurations. These six cases are discussed in LEWIS and SHEDLER [Ref. 11]. Examples of each case are illustrated in Figures 7 through 12 located at the end of this section.

2. Step Two

a. If the intensity function $\lambda(t)$ is monotone increasing or monotone decreasing over the interval (Cases I, II, IV and V; see Figures 7, 8, 10 and 11) the intensity function is decomposed into two separate intensity functions over the same interval. The resulting intensity functions are of the form;

$$\lambda(t) = \exp(\gamma_0 + \gamma_1 t)$$

and

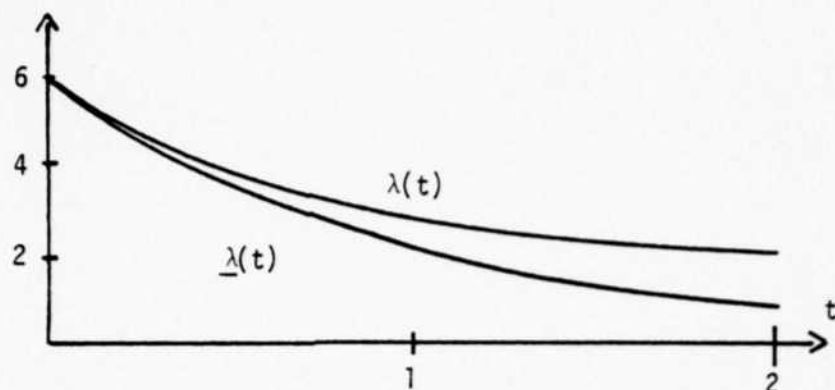
$$\lambda^*(t) = \lambda(t) - \underline{\lambda}(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2) - \exp(\gamma_0 + \gamma_1 t)$$

It is clear that $\underline{\lambda}(t) + \lambda^*(t) = \lambda(t)$.

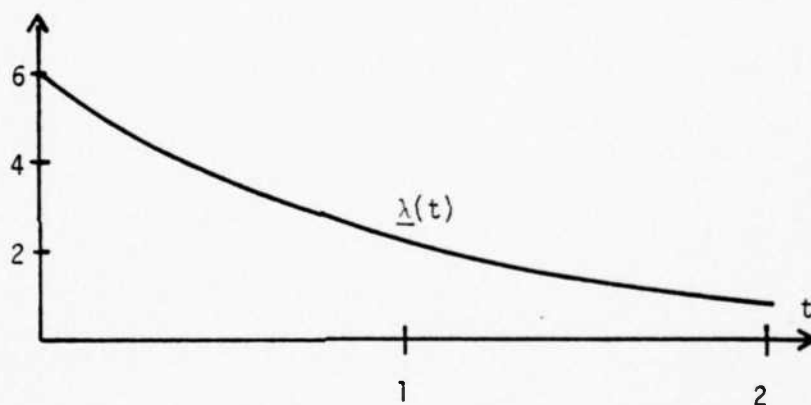
b. If either of the two cases not covered by 2a above occurs, $\lambda(t)$ will be monotone increasing (decreasing) on the subinterval $(0, b]$ and monotone decreasing (increasing) on the complementary subinterval $(b, t_0]$ (see Figures 3 and 6), where b is a unique point within the interval at which $\lambda(t)$ has a maximum (minimum) value. By dividing the interval properly into two disjoint, contiguous subintervals, each subinterval may be treated as explained in 2a. Subsequent steps are applied to each of the two intervals separately and the results combined.

3. Step Three

An efficient algorithm for generating a non-homogeneous Poisson process with a log-linear intensity function (i.e. $\lambda(t) = \exp(\beta_0 + \beta_1 t)$) is presented by LEWIS and SHEDLER [Ref. 13]. This algorithm generates the non-homogeneous Poisson process through the use of gap statistics. (A comparison of the gap statistic technique with the conventional integral transform technique is discussed in Appendix C.) By judicious selection of the coefficients of the log-linear intensity function, most of the total area under the original intensity function $\lambda(t)$ will be contained under the function $\underline{\lambda}(t)$. Therefore most of the events in the non-homogeneous Poisson process with intensity function $\lambda(t)$ can be accounted for by employing the gap statistics algorithm on the intensity function $\underline{\lambda}(t)$.



Step 1 - Categorize intensity function



Step 2 - Decompose $\lambda(t)$ into $\lambdā(t)$ (log-linear) and $\lambda^*(t) = \lambda(t) - \lambdā(t)$.

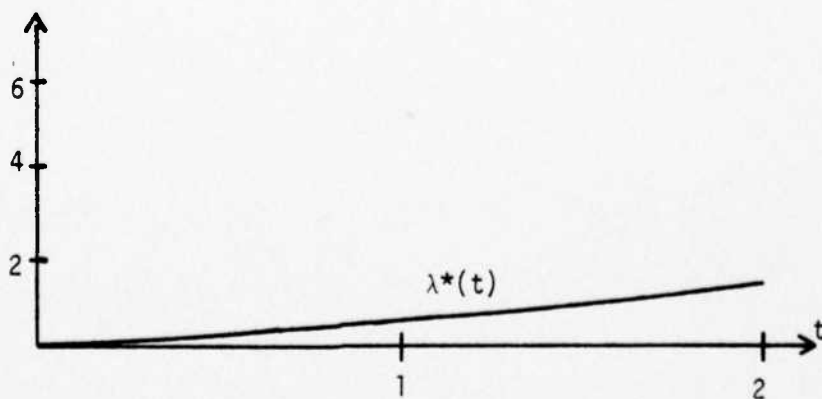
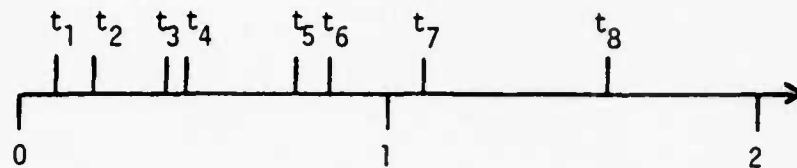


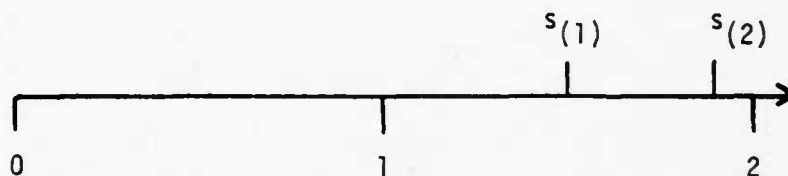
Figure 4 - DIAGRAM OF POISSON-DECOMPOSITION AND GAP STATISTIC ALGORITHM



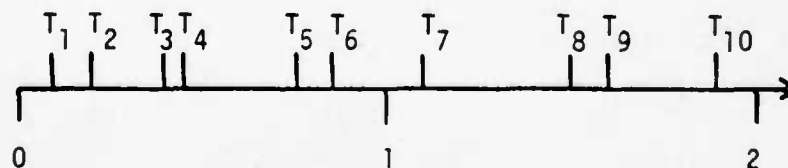
Step 3 - Generate events T_i from log-linear intensity function, $\underline{\lambda}(t)$, using gap statistic algorithm. Events will be ordered.



Step 4 - Generate events s_i from intensity function $\lambda^*(t) = \lambda(t) - \underline{\lambda}(t)$ using rejection algorithm. Events will not be ordered.



Step 5a - Order events from Step 4.



Step 5b - Merge (superpose) events from Steps 3 and 5a. Result is event stream T_1, T_2, \dots from $\lambda(t) = \underline{\lambda}(t) + \lambda^*(t)$.

Figure 5 - DIAGRAM (CONTINUED)

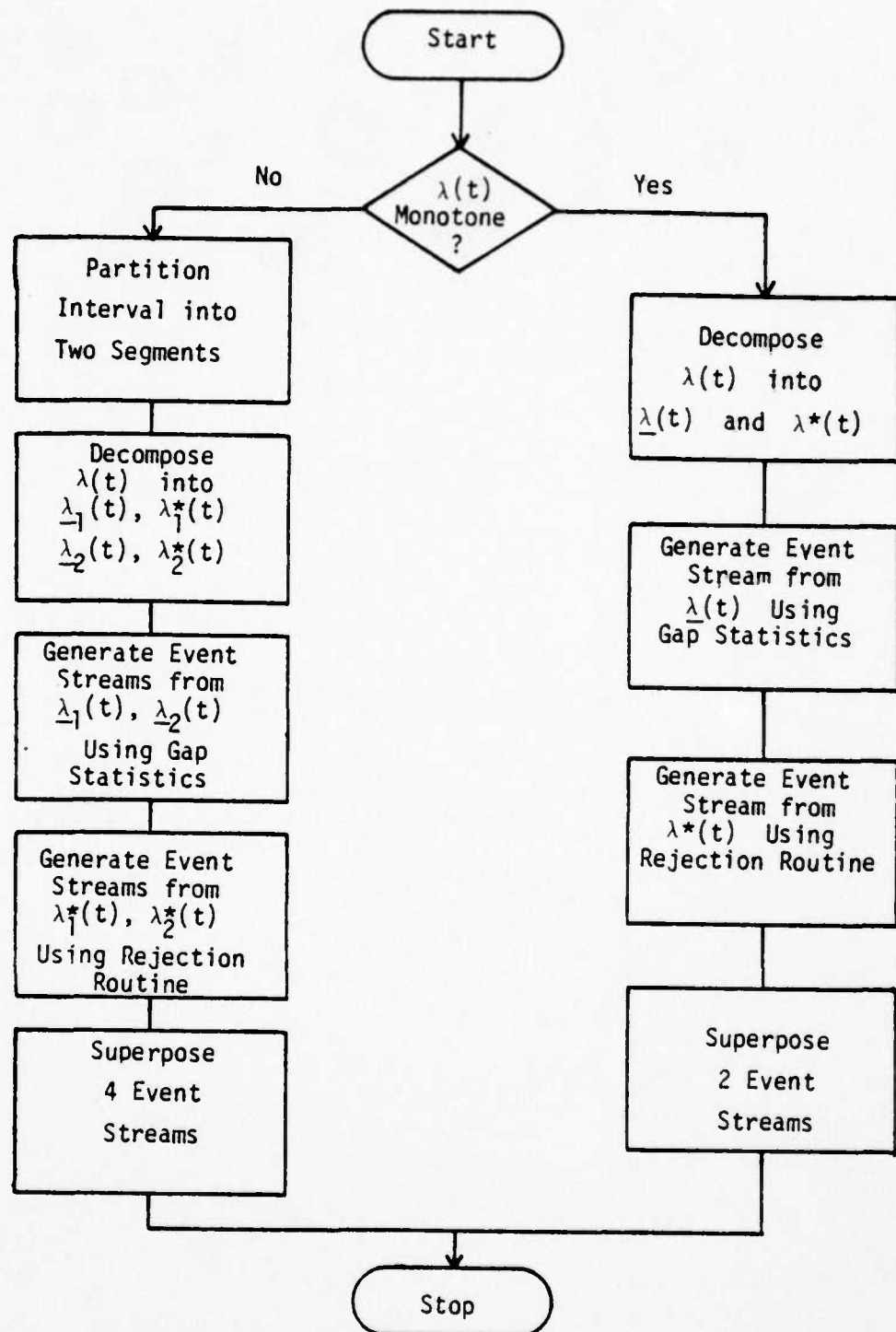


Figure 6 - FLOW DIAGRAM OF POISSON-DECOMPOSITION AND GAP STATISTIC ALGORITHM

4. Step Four

All that remains to be done is to generate an ordered sample of some given size, on the interval $(0, t_0]$, from the remaining component of the original intensity function, i.e. $\lambda^*(t)$.

The number of events in the interval is a Poisson random variable N'_{t_0} with parameter $\mu' = \int_0^{t_0} \lambda^*(t) dt$. Once a realization on this random variable is obtained, i.e. $N'_{t_0} = n'$, Theorem 1 may be invoked. Since $\lambda^*(t)/\mu'_0$ is more easily evaluated than its indefinite integral, the rejection technique (explained in Section IV-D) is used to generate the n' required variates. The rejection technique is not, in general, always an efficient method for variate generation unless great care is taken. Yet in this decomposition scenario, it will be used to generate only a small percent of the total events required by the original intensity function $\lambda(t)$. The majority of the events will be generated by the efficient gap statistics algorithm. The efficiency gains should more than compensate for any efficiency losses due to the use of the rejection technique.

5. Step Five

The events produced by Step 3 will be in order on the interval $(0, t_0]$. The events produced in Step 4 will not be in order on $(0, t_0]$. By ordering the events from Step 4, (which are few in number compared to the total number of events in the non-homogeneous Poisson process) it is possible to superpose the two ordered event streams. The merged event streams produce a new event stream from the original intensity function $\lambda(t)$.

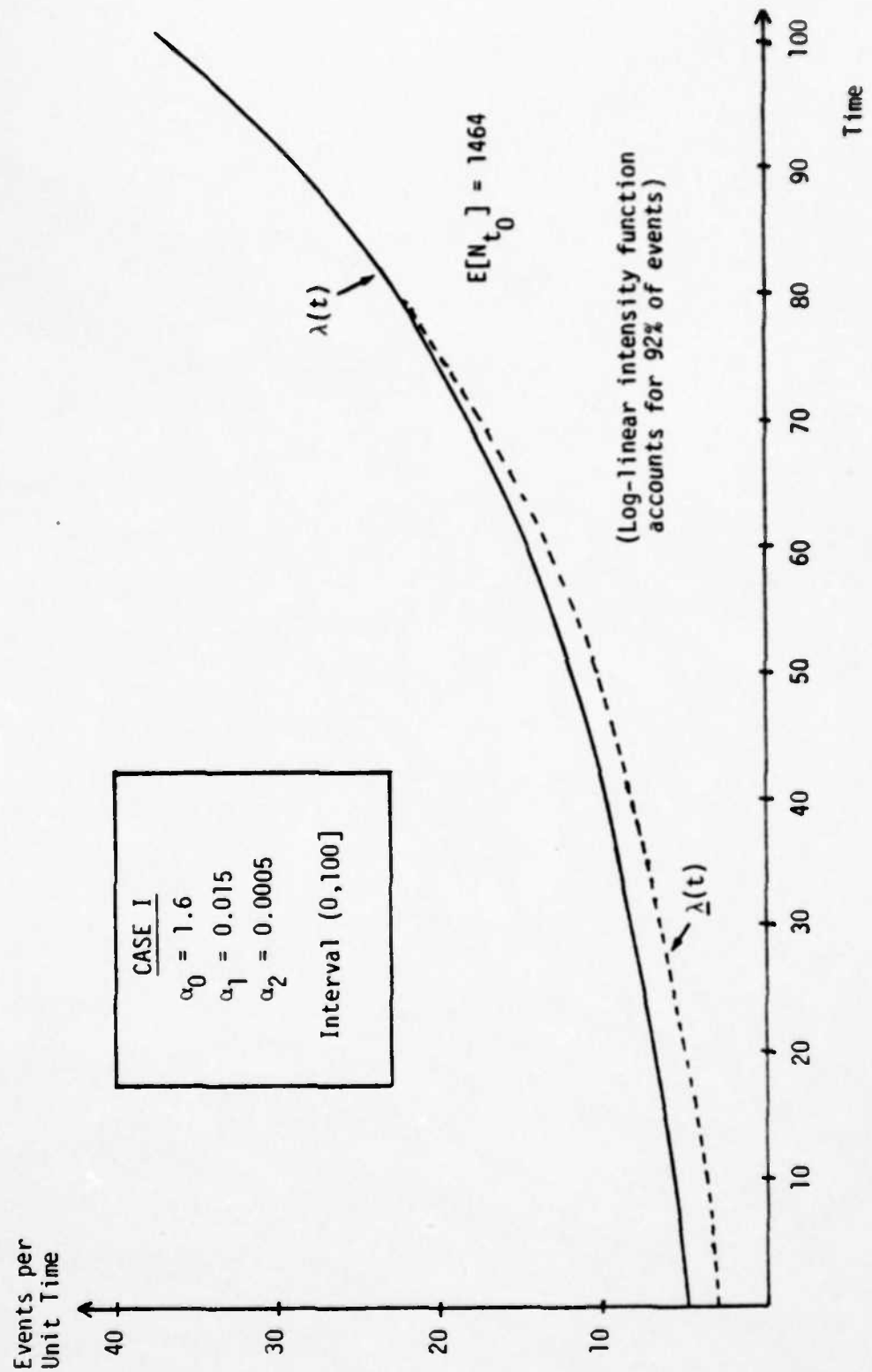


Figure 7 - SAMPLE INTENSITY FUNCTION - CASE I

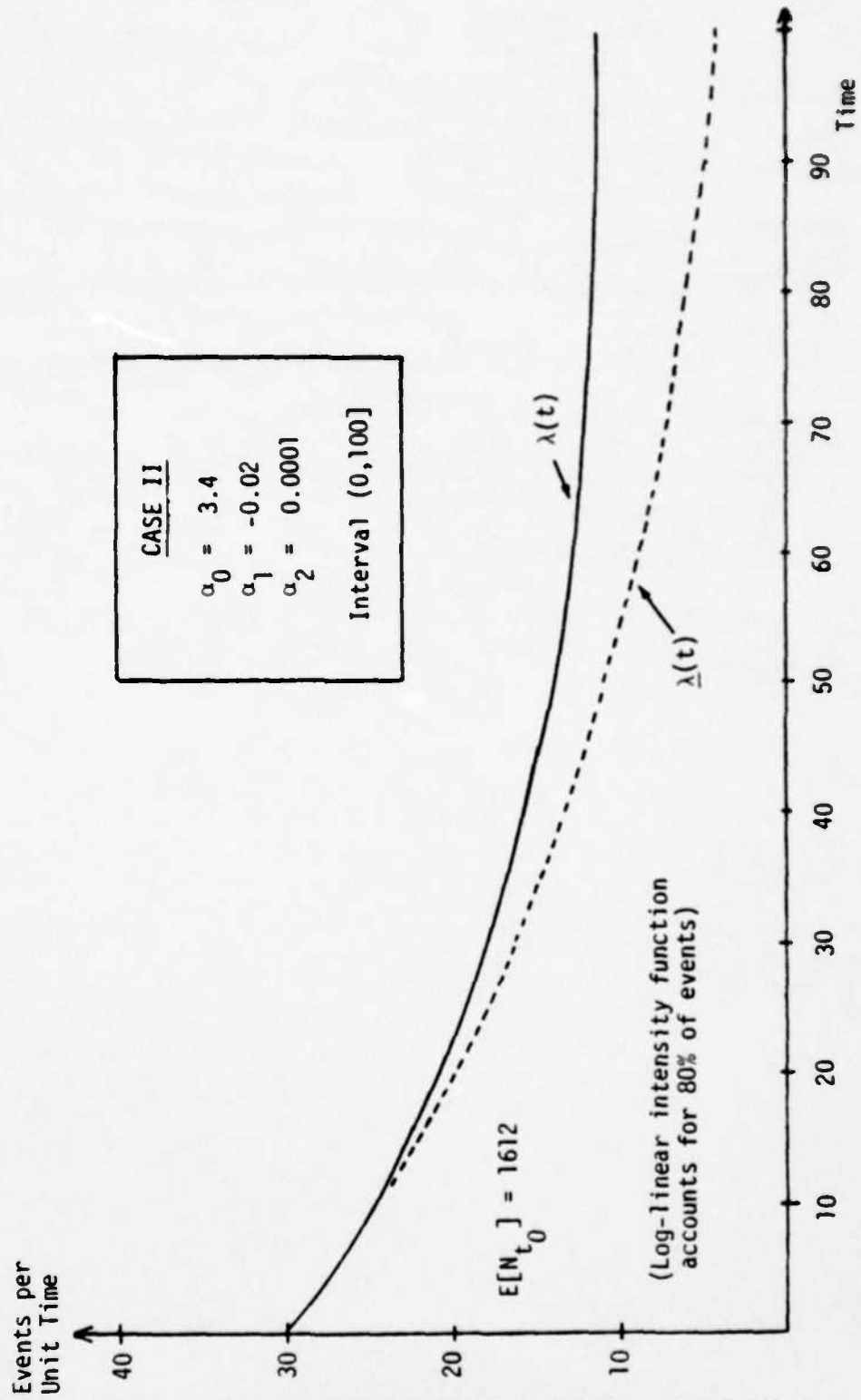


Figure 8 - SAMPLE INTENSITY FUNCTION - CASE II

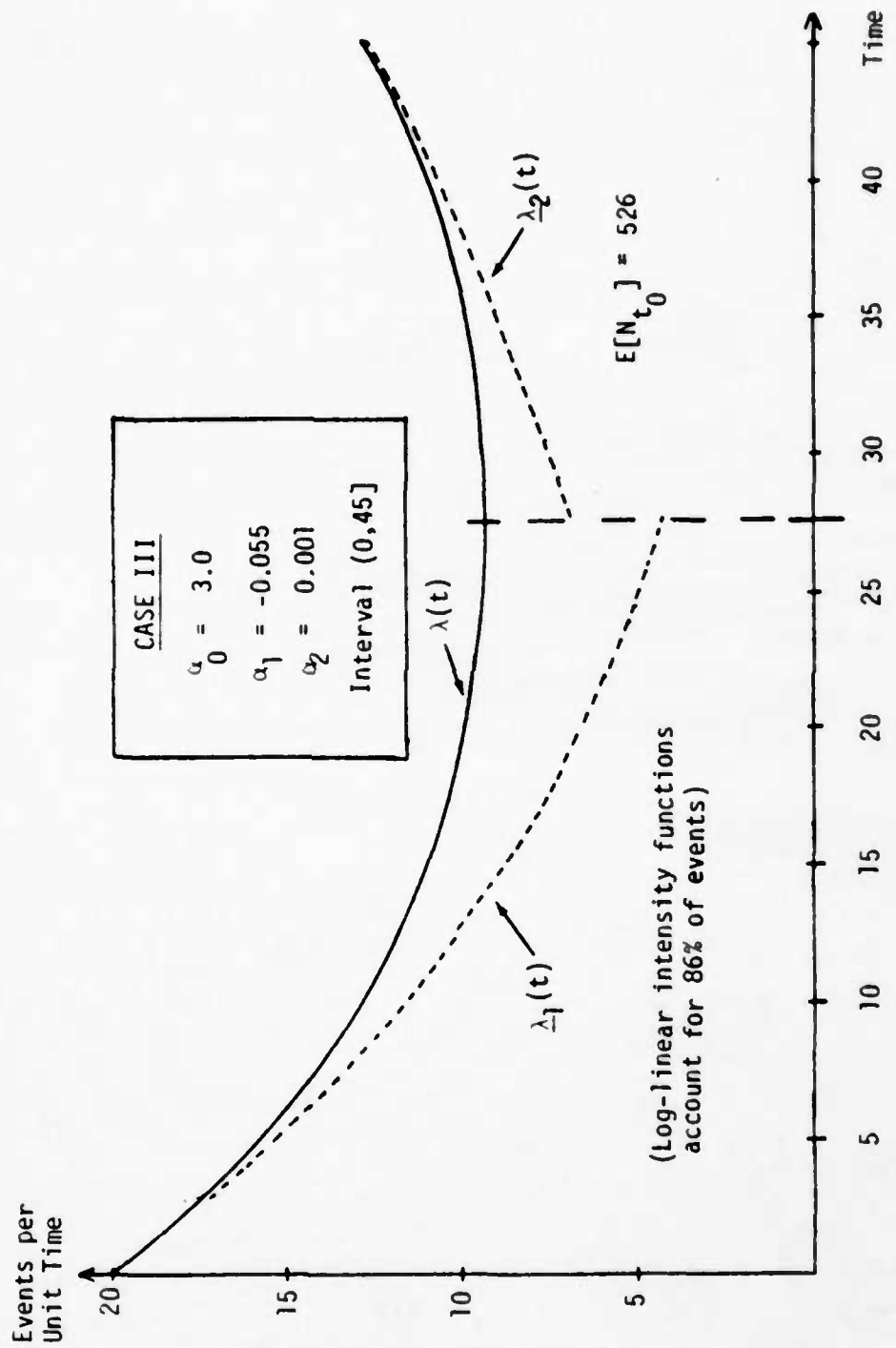


Figure 9 - SAMPLE INTENSITY FUNCTION - CASE III

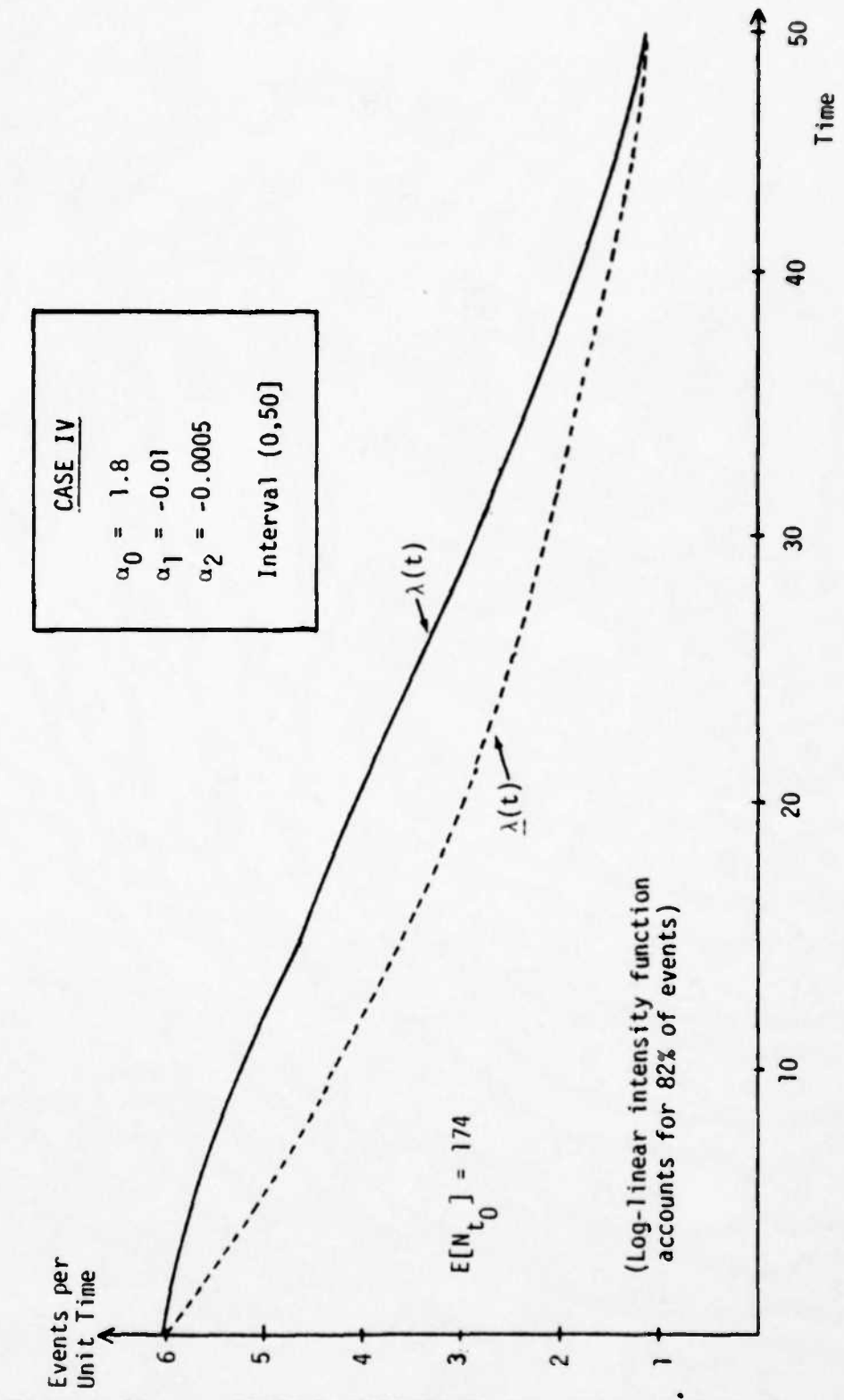


Figure 10 - SAMPLE INTENSITY FUNCTION - CASE IV

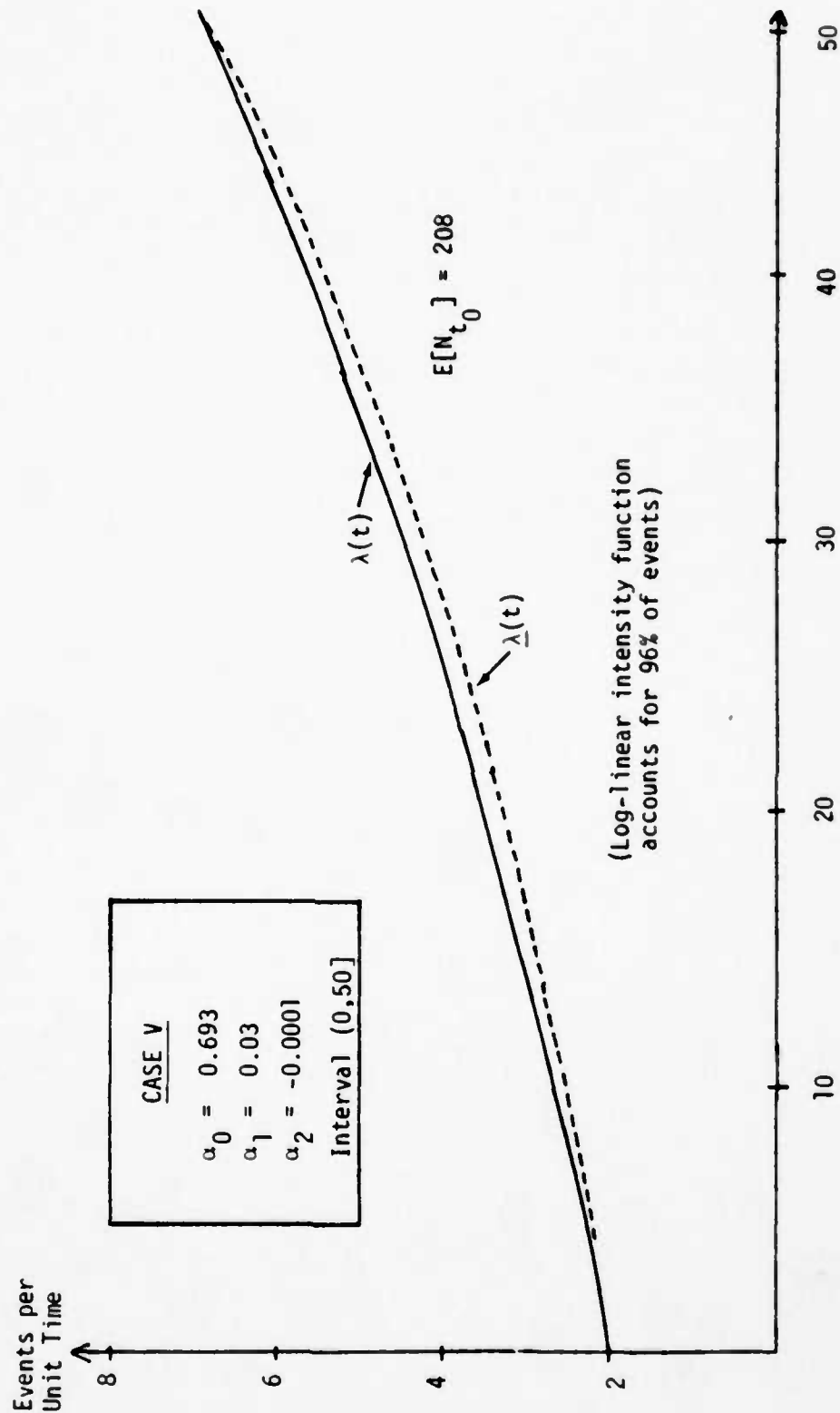


Figure 11 - SAMPLE INTENSITY FUNCTION - CASE V

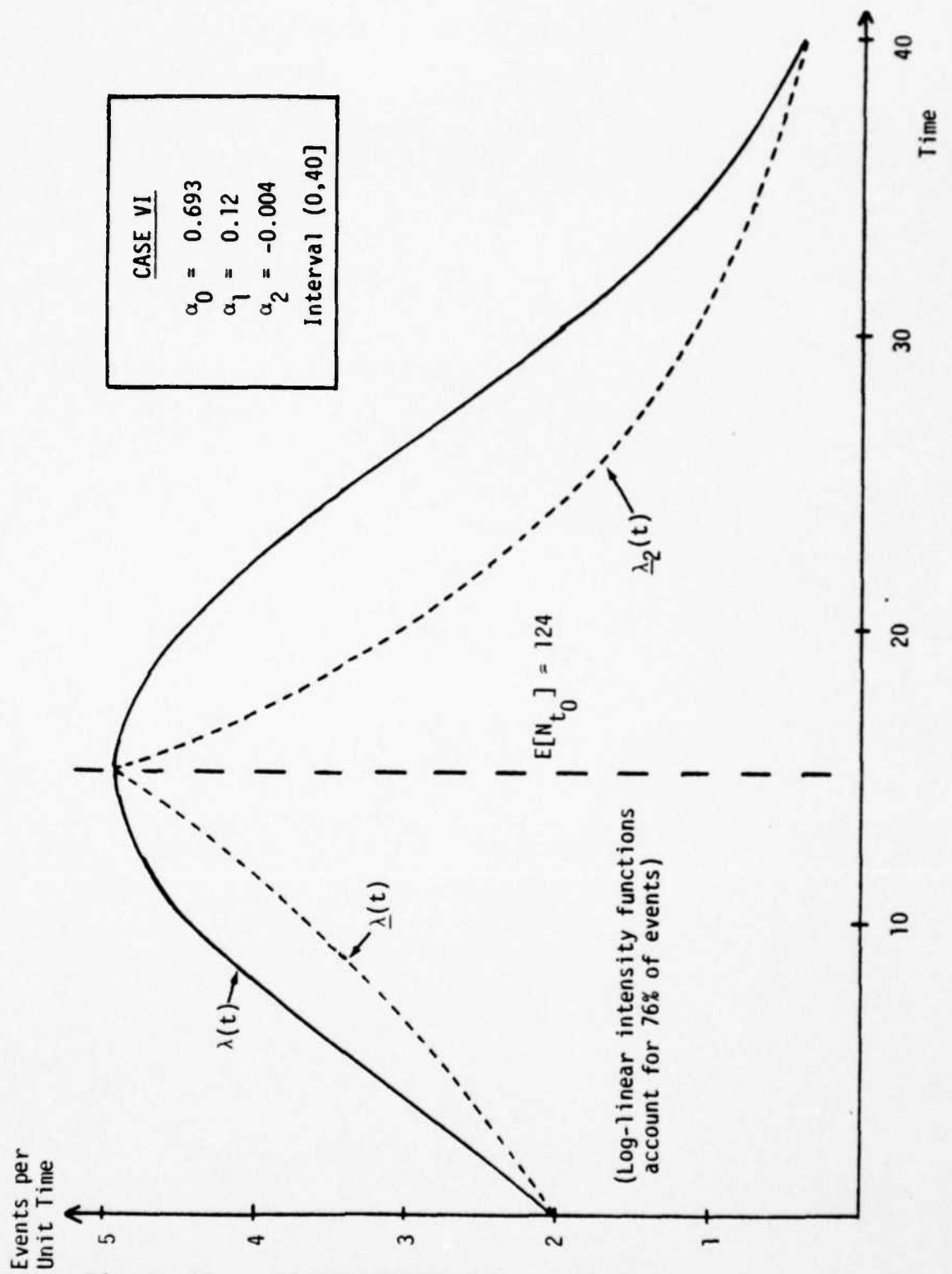


Figure 12 - SAMPLE INTENSITY FUNCTION - CASE VI

IV. ALGORITHM IMPLEMENTATION

A. GENERAL

This section explains the several specific techniques that were applied to implement the two competing algorithms (Algorithm A and Algorithm B) into FORTRAN computer programs. Detailed discussion of the various subprograms is avoided since References 11 and 13 and attached program listings provide such information. The Algorithms A and B have some subprogram requirements in common while other subprograms are unique to one algorithm or the other. This section will discuss first those requirements common to both algorithms, then those needed only by the time-scale transformation algorithm (Algorithm A) and finally those unique to the Poisson-decomposition and gap statistic algorithm (Algorithm B). Hereafter differentiation between the algorithms and the FORTRAN computer program implementation of the algorithms will not always be made. The meaning will be clear from the context.

B. COMMON REQUIREMENTS

1. Integration of a Degree-Two Exponential Polynomial Function over a Fixed Interval

Algorithm A requires that the intensity function

$\lambda(t)$ be integrated over a fixed interval in order to determine the value of the parameter of the Poisson random variable that governs the number of events that are observed to occur in the non-homogeneous Poisson process. Algorithm B requires that the intensity function $\lambda^*(t) = \lambda(t) - \underline{\lambda}(t)$, be integrated over a fixed interval for the same reason. Since

$$\int_a^b \lambda^*(t) dt = \int_a^b \lambda(t) dt - \int_a^b \underline{\lambda}(t) dt$$

and $\underline{\lambda}(t)$ has an explicit expression for its indefinite integral, i.e.

$$\int_a^b \underline{\lambda}(t) dt = \exp(\gamma_0) \left\{ \frac{\exp(\gamma_1 t)}{\gamma_1} \right\},$$

the problem for both algorithms is reduced to computing the value for

$$\int_a^b \lambda(t) dt = \int_a^b \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2) dt.$$

SUBROUTINE HELP employs IMSL FUNCTION MMDAW or the FORTRAN supplied procedure DERF, as appropriate, to perform this calculation. Section III-B and Appendix B discuss how this computation could also be made using a convergent series representation.

2. Generation of a Poisson Variate with a Given Parameter

Both candidate algorithms require at least one realization on a Poisson distributed random variable with a given parameter. The value assumed by the random variable

is the number of events observed in a specific interval of time over which the occurrence of events is governed by a given intensity function (i.e. $\lambda(t)$ for Algorithm A and $\lambda^*(t)$ for Algorithm B). The nature of most real event streams which lend themselves to analysis using a non-homogeneous Poisson process model is that they consist of a large total number of events. This will be the case either if a dense process is observed over an interval of short or moderate length or if a "sparse" process is observed over an extended interval (see the arrivals at an intensive care unit given in LEWIS [Ref. 9]). The point is that both algorithms must be flexible enough to generate non-homogeneous Poisson processes that result in high numbers of events occurring. Since the number of events occurring over any interval in such a process is a Poisson random variable, it becomes necessary to be able to generate a variate from a Poisson distribution with a large parameter, i.e. large mean. The two most theoretically straightforward algorithms for generating Poisson distributed variates (the additive and multiplicative methods) become computationally cumbersome as the parameter of the random variable increases. Both the additive and multiplicative algorithms and the practical deficiencies of each are discussed in Appendix D.

The technique selected to deliver a Poisson variate on demand to both Algorithm A and Algorithm B is the Gamma method. It is developed and explained in an unpublished book by AHRENS and DIETER [Ref. 1]. A paraphrased account of the development of this algorithm is included in Appendix E. The main advantage of AHRENS and DIETER's Gamma method is that whereas the computer time required for the additive and multiplicative methods is proportional to the parameter of the Poisson distribution being sampled, the computer time required by the Gamma method is proportional to the logarithm of the parameter. The SUBROUTINE PVAR

employs the Gamma algorithm to return a Poisson variate when given a parameter as an input.

3. Event Storage

Any algorithm which simulates a non-homogeneous Poisson process must have some mechanism to provide the user with information that completely describes the realization(s) of the process simulated. Since the specific arrangement of the stream of events over the interval is the information of interest to the analyst, the location, or time of occurrence, of each single event on the interval must be stored. Equivalently, the spacings between events would completely describe the realizations on a non-homogeneous Poisson process. Thus event spacing information rather than event location information could be stored. (Programs written for candidate algorithms A and B both provide the option for the user to demand either event location or event spacing information.) When using either algorithm, an array large enough to hold location or spacing data for each event generated by the algorithm must be created. Since the number of events observed in any realization of a non-homogeneous Poisson process is itself a random variable, the precise size of the array cannot be determined a priori. If the programs are to have any value for general application, they must be able to accept intensity function parameters which will demand large numbers of events when simulated. Using the somewhat arbitrary assumption that an event stream with an average of 4500 events is sufficient for most simulation scenarios, a fixed array with a capacity for 5000 events is used in the programs implementing both of the algorithms. If the value of the intensity function integrated over the interval is as high as the maximum limit of 4500 (i.e. the number of events in the interval is Poisson distributed with mean = 4500)

then the array of length 5000 allows the Poisson random variable to exceed its mean by 7.45 standard deviations before an array overflow is encountered. This highly unlikely event is of such rare occurrence (less than one chance in a billion) that it may be disregarded. However, should it occur, the programs will reinitialize themselves and generate a new Poisson process. Also an error indicator is incremented. This error indicator (IER) may be written on demand. Its value is the number of times the program was forced to abort generating a Poisson process, reinitialize itself and start again.

A 5000 element array is small enough to avoid its being an undue memory requirement burden on most operating systems, yet large enough to accommodate most non-homogeneous Poisson processes of interest. Its choice, though arbitrary, was based upon the above two considerations.

C. SPECIAL REQUIREMENTS OF THE TIME-SCALE TRANSFORMATION ALGORITHM (ALGORITHM A)

1. Uniform Variates

As explained in Section III-B, once the number of events observed in the non-homogeneous Poisson process is established (i.e. $N_{t_0} = n$), it is necessary to construct a homogeneous Poisson process consisting of n events over an interval of length μ_0 units. This is easily done by generating n uniformly distributed variates on the interval $(0,1)$ and then scaling each by the factor μ_0 . The LLRANDOM computer library package developed by LEWIS and LEARNORTH [Ref. 12] is a very efficient source of such variates. Once

an array of n uniform $(0,1)$ variates is obtained from LLRANDOM, each element of the array must be multiplied by the appropriate scaling factor. It is these resulting numbers which must be acted upon by the inverse of the integrated intensity function to yield the location of events in the non-homogeneous Poisson process on the original interval $(0, t_0]$.

2. Sorting of Events

To be of much practical value a simulation routine for a non-homogeneous Poisson process should order the events in the interval from first to last. In Algorithm A, this ordering may be done before applying the inverse integrated intensity function to the elements in the homogeneous Poisson process. The monotonic nature of the integrated intensity function maintains the relative order of all elements after they have been transformed by the inverse function (see Figure 3). Of course the ordering could be done after the transformations also. In implementing Algorithm A, the uniform variates were scaled by the factor μ_0 then ordered, from lowest to highest, before being transformed by the inverse of the integrated intensity function.

Ordering of large arrays of numbers is a time consuming operation on the computer. There are many ordering algorithms of varying degrees of sophistication and efficiency. Because ordering is unavoidable when using Algorithm A, selection of an efficient ordering routine is important. The ordering algorithm used in this implementation was the W. R. CHURCH computer center library routine PXSORT which employs Singleton's version of the partition exchange sort. (A program listing of PXSORT is provided in the computer center's catalogue of library

routines.) The PXSORT routine appears to be the most efficient ordering routine readily available for the purpose. (It is acknowledged that more efficient routines specifically tailored to the problem of ordering uniform random variates may possibly have been developed that would improve the overall efficiency of the program implementing Algorithm A.)

3. Computation of the Transformed Values

The essence of Algorithm A is the application of the inverse of the integrated intensity function to each event in the homogeneous Poisson process on the interval $(0, \mu_0]$. As mentioned before, the intensity function, $\lambda(t)$, under consideration does not yield an explicit expression for the integrated intensity function, $\Lambda(t) = F(t)$, that must be inverted. (Note: $F(t)$ is a "scaled" distribution function). Hence neither does a computationally convenient expression for this inverse function exist. Numerical methods must be employed to transform the position of each event on the interval $(0, \mu_0]$ in the homogeneous process to its corresponding position on the interval $(0, t_0]$ over which the simulated non-homogeneous Poisson process is to be produced.

The Newton-Raphson technique was used to accomplish the transformation. This technique allows for iterative approximations which converge to the true transformed values. (i.e. t_1, t_2, \dots, t_n ; where $t_i = F^{-1}(u_i)$) The iterations continue for each value u_i until an estimate t' for its corresponding t_i is found that satisfies $|F(t') - u_i| < \epsilon$, where ϵ is a predetermined tolerance level. (The specific value for ϵ used was $\mu_0 \times 10^{-5}$.)

In the Newton-Raphson technique, given a function $h(x)$, the objective is to find the solution, x^* , satisfying

the equation $h(x^*) = 0$. Letting x_1 be the initial estimate for x^* (based upon some reasonable criteria) the next estimate for x^* , that is, x_2 , can be calculated from the fundamental iteration relationship,

$$x_{k+1} = x_k - \frac{h(x_k)}{h'(x_k)} .$$

This assumes that the function $h(\bullet)$ and its derivative $h'(\bullet)$ can be evaluated at x_k . The process is repeated k times until $|h(x_k)| < \epsilon$; or, until $|x_k - x_{k+1}| < \epsilon$. This procedure is nothing more than using the first two terms of the Taylor series expansion of the function $h(\bullet)$ to locate the x -intercept of the line tangent to $h(x)$ at the point $h(x_k)$. That intercept value becomes the next approximation for the root of $h(x)$ and the procedure is repeated. A graphical explanation of the Newton-Raphson method is presented in Appendix E.

Applying the Newton-Raphson method to the present problem requires some special modifications. The problem is to find a value t_i such that $F(t_i) = u_i$, where u_i is known, i.e. to find $t_i = F^{-1}(u_i)$. Direct application of the inverse $F^{-1}(\bullet)$ is impossible since its functional form defies all but the most abstract mathematical expression. However if a new function $G_i(t) = F(t) - u_i$ is defined and if its root t^* , determined (i.e. a t^* such that $G_i(t^*) = 0$) then $F(t^*) - u_i = 0$, and $F(t^*) = u_i$. Thus $t_i = t^*$ is the value desired.

In order to apply the Newton-Raphson method to the function $G_i(t)$ it must be possible to calculate both $G_i(t)$ and its derivative. Since $G_i(t)$ differs from $F(t)$ only by a constant, the problem reduces to that of evaluating $F(t)$. This has already been done, as previously explained in Section III-B, and merely requires the assistance of

SUBROUTINE HELP. Now $G_i'(t) = F'(t)$, so taking the derivative of $G_i(t)$ returns the intensity function $\lambda(t)$ which is easily evaluated for any t . Clearly, the Newton-Raphson method may be used.

It should be noted here that for each u_i there is a corresponding function $G_i(t)$ on which the Newton-Raphson technique must be used. Since each use of the Newton-Raphson method may require several iterations to arrive at a value t^* which satisfies the tolerance criterion, it is readily evident that for large n , considerable computational effort is required to obtain all the values t_i ($i = 1, \dots, n$). The number of iterations required to arrive at a suitable approximation for each t_i is highly sensitive to the accuracy of the initial approximation (designated $t_{i,1}$). If the initial approximation is close to the actual t_i , the Newton-Raphson method will converge very quickly. If the initial approximation is poor, convergence could be much slower. The procedure used to select initial approximations for each t_i will therefore have a profound effect upon the overall efficiency of Algorithm A.

Selection of these initial approximations is done by partitioning the interval $(0, t_0]$ into equal length segments. The number of segments is equal to $\min[10, n/4]$. The function $F(\cdot)$ is evaluated at the end points of each segment and these end points, with their corresponding function values are stored in an array. This procedure is performed by SUBROUTINE BNCHMK. See Appendix E for a graphical representation.

To find an initial approximation for the t_i which corresponds to any given u_i , the array of function values is searched until two adjacent function values which bracket u_i are located. These adjacent function values uniquely

identify the segment of the interval $(0, t_0]$ in which the elusive t_i is located. Either end point of the segment would serve as a good initial approximation $t_{i,1}$ for the Newton-Raphson method. However, for the purpose of this implementation, the end point which yielded the function value closer to u_i is used for the initial approximation.

The decision to divide the interval $(0, t_0]$ into $\min[10, n/4]$ segments was based upon empirical results. Of several proposed segmenting schemes that one which resulted in the fewest total number of distribution function evaluations over several intensity function/interval scenarios was chosen. Higher values than $n/4$ for sparse event streams may produce better results. But $n/4$ gave good results for dense streams and adequate results for sparse streams. It appeared to be the best compromise as a candidate for general usage. (The number of function evaluations of $G_i(t)$ for each t_i averaged between 2.2 and 2.7 for this partitioning scheme.)

Alternative methods would be to use for the initial approximation for t one of the following values:

- $t_{i-1} \quad (< t_i)$
- $t_{i-1} + t/n$

Neither of these other methods were attempted so it is uncertain whether they would be more or less efficient than the method which was used. Efficiency differences among the three methods are probably not substantial since each will usually yield a good first approximation.

D. SPECIAL REQUIREMENTS OF THE DECOMPOSITION ALGORITHM (ALGORITHM B)

1. Intensity Function Categorization

The decomposition routine selected depends on which of six possible shape categories the intensity function $\lambda(t)$ falls into (c.f. Figures 7 through 12). An examination of the constants α_0 , α_1 and α_2 in the intensity function and the interval $(0, t_0]$ over which the non-homogeneous Poisson process is to occur will uniquely identify the category of the intensity function. A thorough discussion of this procedure is presented in Ref. 11, and is not reproduced here. Implementation of the category identification procedure requires a lengthy sequence of decision statements within the computer program.

2. Selection of the Imbedded Log-Linear Intensity Function(s)

Once the intensity function has been categorized it must be decomposed in accordance with a complex scheme described in Ref. 11. The objective of the decomposition scheme is to fit a log-linear curve (or curves) completely underneath $\lambda(t)$ on the interval (i.e. $\underline{\lambda}(t) \leq \lambda(t)$, $0 \leq t \leq t_0$) in such a way as to maximize the area under the log-linear curve(s). This is done by partitioning of the interval $(0, t_0]$ into subintervals $(0, b]$ and $(b, t_0]$ if necessary, and then by proper selection of the coefficients γ_0 and γ_1 for the log-linear function(s). These coefficients are functions of the coefficients α_0 , α_1 and α_2 in the original

intensity function $\lambda(t)$ and of the interval $(0, t_0]$. As the program advances through the categorizing decision statements, the proper coefficients are computed for the appropriate log-linear function(s).

3. Gap Statistics Algorithm

The precursor to LEWIS and SHEDLER [Ref. 11] was a paper by the same authors [Ref. 13] proposing a gap statistics algorithm for simulating a non-homogeneous Poisson process with a log-linear intensity function $\lambda(s) = \exp(\beta_0 + \beta_1 s)$. (Reference 11 reviews this algorithm in detail.) As previously noted, Algorithm B divides the intensity function $\lambda(t)$ into a sum of two intensity functions, $\underline{\lambda}(t)$ and $\lambda^*(t)$. The new intensity function $\underline{\lambda}(t)$ is chosen to take on the form of a log-linear function so that the gap statistics algorithm may be used to generate a stream of events from this portion of the original intensity function $\lambda(t)$. The SUBROUTINE NHPP2 implements the LEWIS and SHEDLER gap statistics algorithm for the input intensity function $\underline{\lambda}(t)$ and returns an appropriate event stream to the calling program. Since the use of the gap statistics algorithm within Algorithm B is the rationale for suspecting it to be a more efficient algorithm than the time-scale transformation, it is instructive to examine the efficiencies gained in using the gap statistics algorithm vice a time-scale transformation algorithm on a log-linear intensity function. This question is addressed in Appendix C. It was found that use of the gap statistics algorithm resulted in a reduction in computer time of approximately 50% from that required by the time-scale transformation method.

4. The Rejection Routine

The gap statistics algorithm has efficiently produced an event stream from a non-homogeneous Poisson process defined by the log-linear intensity function $\lambda(t) = \exp(\gamma_0 + \gamma_1 t)$. But the process to be simulated has intensity function $\lambda(t) = \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2)$. It is therefore necessary to superpose an event stream from the intensity function

$$\begin{aligned}\lambda^*(t) &= \lambda(t) - \underline{\lambda}(t) \\ &= \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2) - \exp(\gamma_0 + \gamma_1 t)\end{aligned}$$

onto the event stream obtained from the log-linear intensity function.

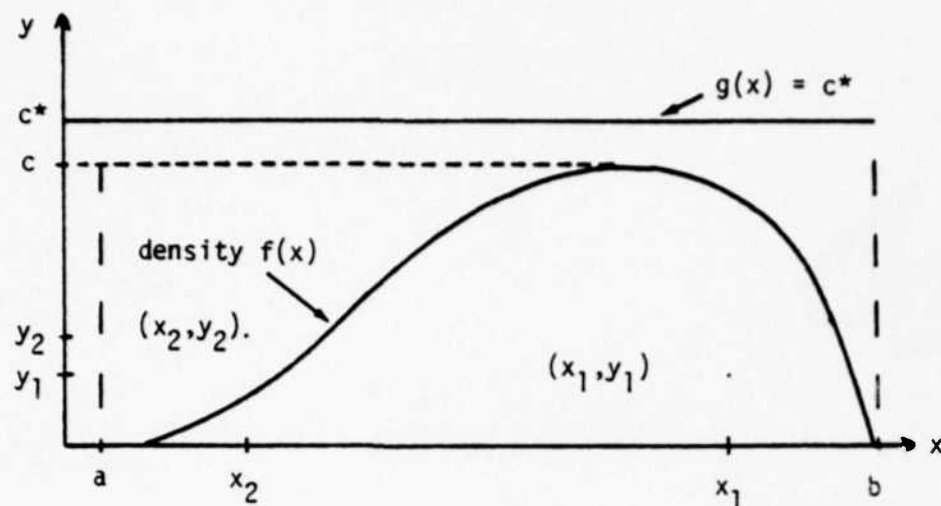
Once it is determined how many events are to occur over an interval with intensity function $\lambda^*(t)$, i.e. $N'_{t_0} = n'$, it is necessary to select an ordered sample of n' variates from a distribution with density function

$$f^*(t) = \frac{\lambda^*(t)}{\int_0^{t_0} \lambda^*(t) dt}$$

The value n' will be small compared to the total number of events in the original non-homogeneous Poisson process. Therefore the need for realizing efficiency in generating these n' ordered variates is not usually as crucial to overall algorithm performance as is the need for efficiency in producing the non-homogeneous Poisson process from the log-linear intensity, $\lambda(t)$, in Step 3. However if the technique for obtaining these n' ordered variates is

extremely inefficient, much or all of the efficiency gains realized from Step 3 above could be lost here. (In fact an example of such a loss of previously gained efficiency is documented in this thesis; see Section VI-B.)

The rejection method is particularly useful for generating random variates from populations with continuous densities that are bounded and which are concentrated on a finite interval; this is the case for $f^*(t)$. The rejection method and an algorithm for its implementation are presented in LEWIS and SHEDLER [Ref. 11]. A geometric argument will suffice to exhibit the principle involved. Consider the density function $f(x)$, for a random variable X on the interval (a,b) in Figure 13. The maximum ordinate of this density is c . If another function $g(x) = c^* \geq c$ is constructed then the density function $f(x)$ is enclosed within the rectangle $(a,0), (b,0), (a,c^*), (b,c^*)$. If a point within this rectangle is selected at random it will fall either under the density function or above it. If it is under the density curve the abscissa of that point is accepted as a variate from the population. If the point lies above the density curve that point is rejected and another point within the rectangle is selected at random. The procedure continues until n' variates have been produced. The random points within the rectangle are easily produced by generating two independent uniform $(0,1)$ variates and scaling them properly resulting in a point (x,y) , where $x = a + u_1(b - a)$ and $y = u_2 \cdot c^*$. Then if $f(x) \geq y$, x is accepted as a variate, otherwise it is not. After n' variates have been obtained, they are ordered to yield an event stream for a process with intensity function $\lambda^*(t)$. SUBROUTINE REJECT employs this method in the program for Algorithm B.



The density $f(x)$ is completely enclosed by the rectangle $(a,0), (b,0), (a,c^*), (b,c^*)$.

Procedure:

1. Generate 2 independent uniform $(0,1]$ variates u_1 and u_2 .
2. Compute: $x = a + bu_1, y = c^*u_2$.
3. Plot (x,y) .
4. If the point (x,y) lies under the density $f(x)$ accept x as a variate; otherwise go to Step 1.

Example: In the graphical example above x_1 would be accepted as a variate whereas x_2 would not be accepted.
(Note: Ideally, $c^* = c$ and a and b correspond to the lateral limits of the density $f(x)$. This minimizes rejection region.)

Figure 13 - THE REJECTION-ACCEPTANCE METHOD OF VARIATE GENERATION FROM AN ARBITRARY DENSITY

Since the probability that a point on the abscissa will not be accepted as a variate is proportional to the area within the rectangle that is not under the density function, it is advantageous to make this area as small as possible. Therefore it is best to set $c^* = c$ and set the values a and b equal to the end points of the interval over which the density function occurs. This is desirable but not necessary. Figure 13 illustrates the more general case where the rectangle is larger than necessary. One may be required to select a $c^* > c$ if c is difficult to determine. Seldom would a and b not coincide with the lateral limits of the density however.

It is obvious from Figure 13 and the preceding discussion of the rejection method of variate generation that it is the "shape" of the density function which is critical to the validity of the method and not the fact that it integrates to one. Therefore any scaled density would preserve the relative shape of the density function. As long as the value c^* is at least as great as the maximum value of the scaled density, the rejection method will yield valid variates. The intensity function $\lambda^*(t)$ may be thought of as a density scaled by the factor $\mu_0' = \int_0^t \lambda^*(t) dt$. Algorithm B uses the intensity function $\lambda^*(t)$ rather than the density function $\lambda^*(t)/\mu_0'$ in the rejection routine. This eliminates a division step for every function evaluation required by the method.

5. Merging Event Streams

The event streams from the intensity functions $\lambda(t)$ and $\lambda^*(t)$ must be merged to produce an event stream from $\lambda(t) = \lambda(t) + \lambda^*(t)$. In the present case there are two event streams (one long and one short) which are already

ordered. This structure allows for superposition rather than a more general (and more time consuming) ordering procedure. To do this merging job SUBROUTINE COLATE is called.

E. SUMMARY

This section has presented a general discussion of the more significant and interesting procedures used to implement Algorithms A and B efficiently in FORTRAN. The reader is referred to LEWIS and SCHEDLER [Ref. 11] for a detailed listing of steps for Algorithm B. Program listings may also be consulted. These may be found after Appendix E.

V. METHOD OF COMPARISON OF ALGORITHMS

A. GENERAL

Conclusions drawn from the results of comparing the efficiencies of two computer programs are only as sound as the measures of effectiveness upon which they are based. These measures of effectiveness are always somewhat arbitrary, so their selection should be based upon compelling reasoning.

Even after reasonable methods of effectiveness have been defined, only a finite number of trials (usually a small finite number) for a given set of circumstances may be run. Therefore, general statements such as "this algorithm is better than that algorithm" can seldom be made with complete confidence. Yet if the incomplete information gathered reveals certain trends, generalizations hypothesized from such trends may warrant a high degree of confidence.

This section describes how the two algorithms are compared and the rationale for choosing the method employed.

B. MEASURES OF EFFECTIVENESS

1. Computational Speed

Perhaps the most popular efficiency measure for computer programs is their speed of execution. Speed is a natural standard as long as computer time remains a costly, scarce resource.

Algorithms A and B were compared in terms of the mean time required to generate event streams from a given set of intensity functions. It should be noted that since the number of events in each event stream is a random variable, the time required to generate one event stream is also a random variable.

Several event streams with different intensity functions, each having a different expected number of events, are produced. Event streams for each of these intensity functions are replicated many times by each algorithm. The total execution time required for all replications is the "speed" measure of effectiveness.

The number of replications varies with each intensity function. For example, an event stream with an expected number of events less than 200 was replicated 100 times whereas event streams with over 1000 expected events were replicated only 30 times. In both cases the total number of events produced was of the same order of magnitude. A priori it seemed reasonable to assume that computer time required to execute each algorithm's program would be roughly proportional to the total number of events

generated. Therefore in designing the experiment the product of the expected number of events times the number of replications ($E[N_{t_0}] \times \# \text{ reps}$) was kept roughly constant in order to keep demands on computer time reasonably under control. (Note: Program execution times were isolated from compilation and linkage times for the purpose of the computational speed comparison.)

2. Computer Memory Requirements

A second natural efficiency measure for computer programs is their core memory requirement. Both programs were written with the goal in mind of conserving as much memory as possible without unduly increasing program complexity. Core requirements reductions could most likely be accomplished in both programs through more sophisticated programming methods. Therefore this comparison has a major weakness as a measure of effectiveness. Recognizing this caveat, memory requirements for each program were compared.

3. Fidelity

The question of paramount importance is: How well does the program simulate the true process? This is a difficult question to answer when dealing with a finite number of realizations on a process whose theoretical properties can be validated only after an infinite number of realizations.

Several methods of comparison are possible. The simplest is that of comparing the sample means and sample variances of the random variable whose realizations are the number of events generated on an interval by a given intensity function. This random variable should be Poisson

distributed with mean equal to the integral of the intensity function over the interval. For a Poisson random variable, the variance equals the mean. This test, although useful for determining if either algorithm varies grossly from expected performance, yields no information about how the algorithms are distributing their simulated events over the interval. It is necessary to look at discrete segments of the interval and examine the distribution of the number of events produced in each segment by our simulated non-homogeneous Poisson process.

From the definition of a non-homogeneous Poisson process (see Section II-B) we know that the number of events observed over any segment of the interval must be a Poisson random variable with parameter equal to the integral of the intensity function evaluated over the interval. By dividing into several segments the interval over which the Poisson process is being simulated, the number of events in each segment can be observed. Two hypothesis tests may then be made for each segment.

The first test is the dispersion test [Ref. 3, p. 158]. This test seeks to ascertain if the number of events observed over each segment is distributed as a Poisson random variable. Let n_1, n_2, \dots, n_k be k observations on the number N_p of events occurring in a given fixed segment p , and let $\bar{n}_p = (n_1 + \dots + n_k)/k$. If N_p is a Poisson distributed random variable, then the statistic

$$d_p = \frac{\sum_{i=1}^k (n_{p,i} - \bar{n}_p)^2}{\bar{n}_p}$$

has a distribution which is approximated by a chi-squared distribution with $k - 1$ degrees of freedom. If the interval has been partitioned into m segments, then by simulating the non-homogeneous Poisson process k times, we can perform m

hypothesis tests at a selected significance level α . The test statistic d_p would then be compared to the $(1-\alpha)$ percentile of the chi-squared distribution with $k - 1$ degrees of freedom. For a large number of hypothesis tests, say 1000, we would expect approximately 1000α rejections of the null hypothesis, if in fact the N_p 's are Poisson distributed.

For example consider the non-homogeneous Poisson process over the interval $(0, 100]$. If this interval were partitioned into 100 unit segments the number of events occurring in each segment should be Poisson distributed. If 51 event streams are simulated over the interval, a value for the test statistic d may be computed for each segment, i.e.

$$d_p = \frac{\sum_{i=1}^{51} (n_{p,i} - \bar{n}_p)^2}{\bar{n}_p}$$

Assuming a significance level $\alpha = .05$ and comparing each d_p to the critical value $\chi_{50, \alpha}^2 = 67.5048$, we would count the number of test statistics such that $d_p > 67.5048$. If the number of events in the segments are indeed Poisson random variables we would expect, on the average, that 5 out of the 100 d_p values would exceed the critical value.

In the above discussion of the dispersion test, no mention was made of the parameters of the Poisson random variables (assuming that they are Poisson). This is because the dispersion test for Poisson distributions is parameter independent. Thus it is necessary to perform a second hypothesis test to determine if the mean number of events per segment is equal to the difference of the integrated intensity function evaluated at the right and left end points of the segment respectively. For large k the sample mean n is normally distributed. Under the null hypothesis,

i.e. that the mean of $N_p = \int_{a_i}^{a_j} \lambda(t) dt = \mu_p$ (where a_i and a_j are the end points of segment p) n is normally distributed with mean μ_p and variance μ_p/k . The test statistic is

$$c_p = \frac{|\bar{n}_p - \mu_p|}{\sqrt{\mu_p/k}}$$

and the critical value is the $(1-\frac{\alpha}{2})$ percentile of the normal distribution, $z_{\alpha/2}$. Again, there would be one hypothesis test for each segment and if we made 1000 such tests we would expect to reject the null hypothesis approximately 1000α times, on the average.

Continuing the example above, it is now of interest to see if the mean number of events on each segment agrees with the theoretical value determined by the integrated intensity function. This time assuming a significance level of $\alpha = .10$, the critical value for each test statistic c_p would be $z_{\alpha/2} = 1.645$. Each $c_p > 1.645$ would be counted and if in fact the random variables have the hypothesized means, we would expect (on the average) that approximately 10 test statistics out of 100 would exceed 1.645.

After the dispersion test and hypothesis test for the means have been performed on event streams from each algorithm, the test results may be compared to see if either algorithm appears to have null hypothesis rejection percentages which are not consistent with the α levels of the hypothesis tests.

(It should be noted that during the development of the programs histograms of the event streams were plotted to determine whether or not they "fit" their respective intensity functions. Both programs seemed to produce satisfactory results by this subjective measure.)

C. OTHER CONSIDERATIONS

1. Intensity Function Category

Algorithm B classifies the input intensity function into one of six categories and then determines what action to take to generate the appropriate event stream. It might be expected that the speed of Algorithm B will be affected not only by the total number of events to be generated but also by the form of the input intensity function. Decomposition of the intensity function into four sections will probably require more time than if it must be divided into just two sections.

Algorithm A treats all intensity functions alike. Therefore, it is important to compare the two algorithms for all six possible categories of intensity functions. It is possible that Algorithm B could be faster than Algorithm A for one category of intensity function and slower than Algorithm A for a different category of intensity function. The six intensity functions selected for the comparison represent all of the categories defined by Algorithm B, and are, in fact, those intensity functions illustrated in Figures 7 through 12 of Section III. These categories are numbered from I through VI and correspond directly to those listed in Ref. 11.

2. Evaluation of c^* Value in Rejection Routine

The importance of reducing the size of the rectangle enclosing the intensity function $\lambda^*(t)$ prior to beginning the rejection algorithm was mentioned in Section IV-D. If possible c^* should correspond to the maximum value of $\lambda^*(t)$

on the interval $(0, t_0]$. For Cases I and II (Figures 7 and 8) this maximum value is easily determined since it occurs at one of the end points of the interval. For Case III (Figure 9) the maximum values of $\lambda_1^*(t)$ and $\lambda_2^*(t)$ both occur at the point b where the interval $(0, t_0]$ has been partitioned into two subintervals $(0, b]$ and $(b, t_0]$. For Cases IV, V and VI, (Figures 10, 11 and 12) the maximum value of $\lambda^*(t)$ is not so easily determined. LEWIS and SHEDLER [Ref. 11, p. 11] state that it is not possible to find this maximum explicitly. However an upper bound may be determined from the values k and \underline{k} where $\lambda(t) \leq k$ and $\underline{\lambda}(t) \geq \underline{k}$ on the interval $(0, t_0]$ (or subintervals $(0, b]$ and $(b, t_0]$ for Case VI). Then by setting $c^* = k - \underline{k}$ it is certain that $\lambda^*(t) \leq c^*$ on the interval or subintervals of interest. Figure 14 which illustrates the rejection routine for the intensity functions for Case V and VI used in the analysis, reveals that the c^* computed in such a manner may not be very efficient as an upper bound for the rejection algorithm. One would expect some reduction in speed efficiency for Algorithm B when c^* greatly exceeds the maximum value of $\lambda^*(t)$.

3. Designated Tolerance Level

Algorithm A requires selection of a tolerance level which determines the stopping criterion for Newton-Raphson iterations. The speed of the algorithm could be expected to be very sensitive to this tolerance limit. By setting it too small the comparison would be prejudiced in favor of Algorithm B. By not setting it small enough the fidelity of the resulting event stream to the true intensity function would be impaired.

It was determined that a tolerance level of $E[N_{t_0}] \times 10^{-7}$ occasionally demanded that the computer discriminate beyond its significant digit capability in

single precision. This resulted in the failure of the Newton-Raphson iterations to converge, although in theory they would have done so if enough precision had been retained. A tolerance level of $E[N_{t_0}] \times 10^{-6}$ seemed to be unnecessarily demanding on Algorithm A even though the computer could discriminate at this level. A stopping rule fixing $\epsilon = E[N_{t_0}] \times 10^{-5}$ was the compromise that insured reasonable accuracy of the simulated event stream without unfairly burdening Algorithm A.

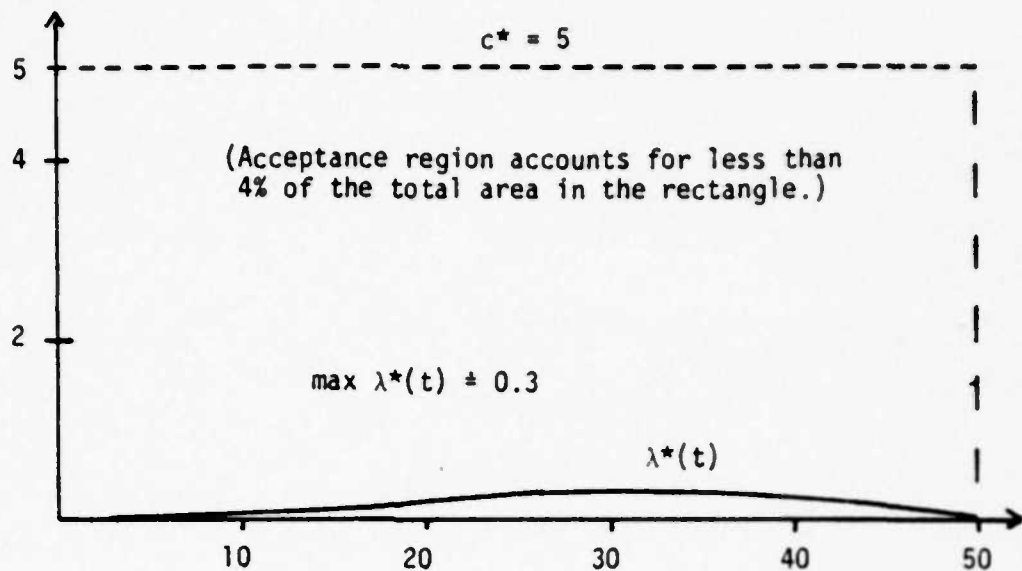


Figure 14a - Graph of Rejection Routine for the Case V
Intensity Function Analyzed

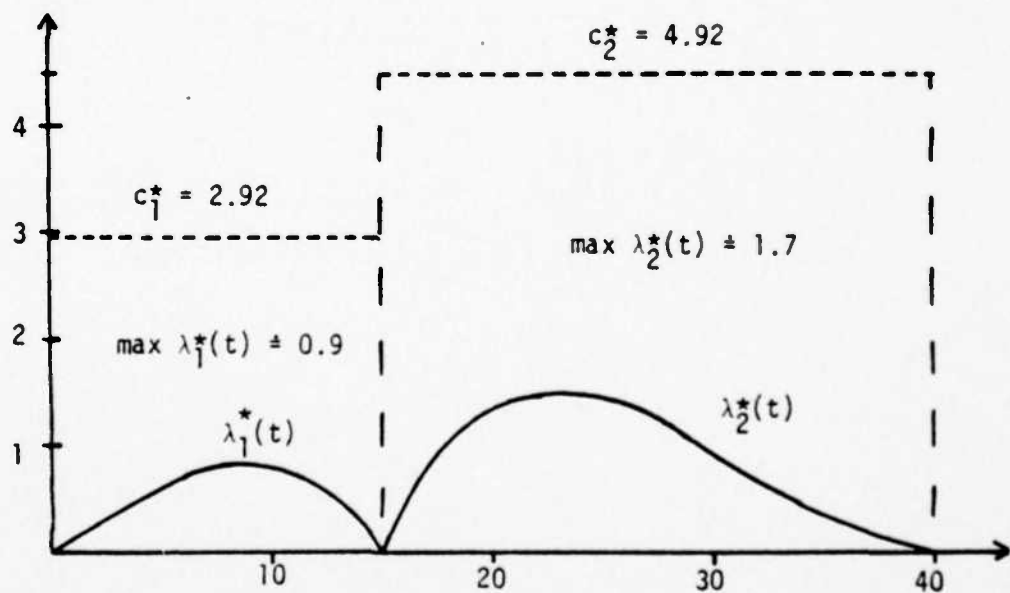


Figure 14b - Graph of Rejection Routines for the Case VI
Intensity Function Analyzed

Figure 14 - ILLUSTRATION OF REJECTION-ACCEPTANCE REGIONS
FOR SAMPLE CASE V AND CASE VI INTENSITY FUNCTIONS

VI. RESULTS AND CONCLUSIONS

A. GENERAL

Computer programs implementing Algorithms A and B were compared in terms of efficiency as explained in Section V. This section documents the results of the comparison and discusses general observations of the author concerning the two competing programs. Also recommendations for further study concerning Algorithm B are offered.

B. MEASURES OF EFFECTIVENESS RESULTS

1. Speed

Table I convincingly illustrates the superiority of Algorithm B over Algorithm A when computational speed is used as a measure of effectiveness. The column in Table I labeled "Time A/Time B" is the ratio of the total time required by Algorithm A to produce a specified number of replications of the given non-homogeneous Poisson process, to the time required by Algorithm B to produce the same number of replications of the process. For all six representative intensity functions the Poisson-decomposition and gap statistic algorithm is at least twice as fast as the time-scale transformation. For large event streams the superiority of Algorithm B is even more pronounced.

Table I - COMPUTATION TIMES FOR EVENT STREAMS
(IBM System 360/67, FORTRAN IV (G) compiler)

CASE	$E[N_{t_0}]$	Reps	Algorithm	\bar{N}_{t_0}	S_N^2	Total Execution Time (sec)	Time A Time B	Discussion
I	1464	30	A ----- B	1464 ----- 1465	1018 ----- 1655	91.5 ----- 12.0	7.60	Best Relative Performance of B over A. Result of $c^* = \max \lambda^*(t)$; dense event stream; 8% use of rejection
II	1612	30	A ----- B	1621 ----- 1620	2110 ----- 1450	99.0 ----- 18.7	5.29	$c^* = \max \lambda^*(t)$; dense event stream; 20% use of rejection routine
III	526	75	A ----- B	526 ----- 528	492 ----- 458	99.1 ----- 19.5	5.08	$c^* = \max \lambda^*(t)$; 14% use of rejection routine
IV	174	100	A ----- B	173 ----- 175	231 ----- 204	36.8 ----- 16.5 (9.6) [#]	2.23 (3.85) [#]	$c^* > \max \lambda^*(t)$; 2nd trial with $c^* = \max \lambda^*(t)$ improved results; 18% use of rejection routine
V	208	100	A ----- B	204 ----- 209	144 ----- 237	48.6 ----- 19.4 (7.2) [#]	2.51 (6.71) [#]	$c^* > \max \lambda^*(t)$; 2nd trial with $c^* = \max \lambda^*(t)$ improved results; 4% use of rejection routine
VI	124	100	A ----- B	124 ----- 124	155 ----- 108	28.4 ----- 11.3 (9.02) [#]	2.51 (3.15) [#]	$c^* > \max \lambda^*(t)$; 2nd trial with $c^* = \max \lambda^*(t)$ improved results; 24% use of rejection routine

[#]Results obtained for 2nd trial with alternate c^* value determined by graphical analysis.

The most obvious disparity revealed in Table I is the difference in speed efficiency (of Algorithm B with respect to Algorithm A) between the group of Cases I, II and III; and the group of Cases IV, V and VI. In the former group Algorithm B was 5 to 7.6 times as fast as Algorithm A. For the latter group Algorithm B was only 2.2 to 2.5 times as fast as Algorithm A.

This difference between the two groups immediately made suspect the value c^* used in the rejection routine of Algorithm B. For the first group c^* is set at the least upper bound for $\lambda^*(t)$. For the second group c^* is not, in general, a least upper bound for $\lambda^*(t)$ but is only an upper bound. To determine whether or not this difference in the "quality" of the c^* values could have such an adverse effect on time efficiency for Cases IV, V and VI, these cases were simulated a second time with new c^* values which were empirically evaluated by graphical methods. These modified c^* values were set close to the least upper bounds of the respective $\lambda^*(t)$ component of the intensity functions for each case. Marked time efficiency gains were observed (see Discussion column, Table I) indicating that significant efficiency losses can be sustained due to the method of setting c^* values in Cases IV, V and VI.

Three factors appear to influence the degree of speed efficiency gains of Algorithm B over Algorithm A. First is the selection of the c^* value as discussed above. The second factor is the percentage of total events which Algorithm B must generate from the rejection routine. For the specific intensity functions analyzed, this percentage ranged from 4% in Case V to 24% in Case VI. The fewer events required from the rejection algorithm relative to the number generated from the gap statistics algorithm, the greater the efficiency of Algorithm B.

Finally the total number of events in an event stream affects relative efficiencies between Algorithms A and B. This is because as the number of events to be sorted by Algorithm A grows, the less efficient its sorting routine becomes. Sorting requirements for Algorithm B are greatly reduced due to the properties of the gap statistic procedure used. Therefore as total events increase, so does the efficiency advantage of Algorithm B over Algorithm A increase.

2. Computer Memory Requirements

Algorithm A requires approximately 72,000 bytes of core storage. Algorithm B demands about 92,000 bytes. Both programs are costly in terms of storage due to the number of library routines used and the need to store up to 5000 event times. The difference of 20,000 bytes would not be important unless computer capacity were being challenged. Memory requirements for both algorithms could be reduced by reducing the event stream capacity of the programs. A reduction of capacity from a maximum of 5000 events to a maximum of 2000 events in each program would result in storage requirements of 54,000 bytes and 68,000 bytes for Algorithm A and Algorithm B respectively.

3. Fidelity

Table II lists the results of six series of hypothesis tests of the types discussed in Section V-B. Three series of dispersion tests and three series of hypothesis tests for the mean were conducted. Intensity functions for Cases III, IV and VI were selected for these tests.

For each of the three cases, a large number (500, 900 or 1000) of each type of hypothesis test was performed for a fixed level of significance α . The number of times each null hypothesis was rejected was recorded and the statistic $\hat{\alpha} = \{(\text{number of rejections})/(\text{total tests})\}$ was computed for the series of dispersion tests and the series of hypothesis tests for the mean. If the null hypothesis is true, then $\hat{\alpha}$ is an estimate of the significance level α .

From Table II it appears that for the hypotheses tests associated with Case III and Case IV, $\hat{\alpha}$ gives a very good estimate for α , the true significance level. For Case VI some disparities are observed. For Algorithm B the $\hat{\alpha}$ value for the dispersion test (i.e. hypothesis test that variates are Poisson distributed) is 0.039 whereas the significance level is $\alpha = 0.01$. Algorithm A yields a much better $\hat{\alpha}$ value of 0.012. For the hypothesis tests for the mean, both Algorithm A and B yield $\hat{\alpha}$ values almost twice that of α .

These results, though interesting, are inconclusive. For the dispersion test, the statistic

$$d_p = \frac{\sum_{i=1}^k (n_{p,i} - \bar{n}_p)^2}{\bar{n}_p}$$

is only approximately distributed as a chi-squared random variable. It has been shown by FISHER [Ref. 4] that for Poisson random variables with low expectations, hypothesis tests based on the chi-squared distribution may produce spurious results. Also, we would expect that if the distribution of d_p is only approximated by the chi-squared distribution, that the relative error between the approximate and true distribution is greatest in the tails, i.e. at high percentile values. Case VI was tested at an α level of 0.01. It is also the intensity function with the

fewest expected number of events of the three intensity functions tested. Partitioning of the interval into segments produced some segments which had a low expected number of events (see the right tail of the density function pictured in Figure 12). The results for Case VI rather than indicating any serious flaws in either of the algorithms suggest further research into finding better ways of hypothesis testing for sparse event streams and highly discriminating levels of significance.

Results of the hypotheses tests on Cases III and IV indicate that both algorithms simulate event streams that conform to the desired non-homogeneous Poisson processes.

Table II - RESULTS OF HYPOTHESES TESTS FOR FIDELITY
(IBM System 360/67, FORTRAN IV (G) compiler)

	Hypothesis: Variates Are Poisson Distributed			Hypothesis: Variates Have $\text{Mean} = \int_a^b \lambda(t) dt = \nu_p$			Accept H_0	Reject H_0
	Case III $\alpha = .10$ Trials: 900	Case IV $\alpha = .05$ Trials: 500	Case VI $\alpha = .01$ Trials: 1000	Case III $\alpha = .05$ Trials: 900	Case IV $\alpha = .10$ Trials: 500	Case VI $\alpha = .01$ Trials: 1000		
ALGORITHM A (Time-Scale Transform)	804	475	988	848	450	983		
	96	25	12	52	50	17		
	$\hat{\alpha} = .107$	$\hat{\alpha} = .05$	$\hat{\alpha} = .012$	$\hat{\alpha} = .058$	$\hat{\alpha} = .10$	$\hat{\alpha} = .017$		
ALGORITHM B (Poisson Decomposition)	814	476	961	842	461	980		
	86	24	39	58	39	20		
	$\hat{\alpha} = .096$	$\hat{\alpha} = .048$	$\hat{\alpha} = .039$	$\hat{\alpha} = .064$	$\hat{\alpha} = .078$	$\hat{\alpha} = .020$		

C. GENERAL OBSERVATIONS

1. Programming Ease

The programming of Algorithm A is simpler than the programming of Algorithm B. Because the time-scale transformation technique handles all intensity functions alike, several different cases need not be identified. Algorithm B must categorize the input intensity function into one of six categories and then must treat each category in a unique way. A rough indication of this difference in the degree of complexity is the number of instruction statements required by each program. The program implementing Algorithm A required 142 instructions. The program for Algorithm B required 364 instructions.

2. Exact Method Versus Approximate Method

Algorithm B employs an exact method in generating the non-homogeneous Poisson event stream. It is exact in the sense that all event times are calculated directly and the precision of those calculations are limited only by the physical constraints of the computer.

Algorithm A employs a Newton-Raphson iterative method to approximate event times on the interval. The stopping criterion for the technique is limited by machine precision considerations. Also the stopping rule does not give a firm account of the magnitude of error that can be expected in the event times. The epsilon criterion is applied to the value of the integrated intensity function

and not to the abscissa, (or time interval axis). For the integrated intensity function, $F(t)$, it is conjectured that given a function value $F(t_i) = u_i$, if a $t = t'$ can be found such that $|F(t') - F(t_i)| \leq \epsilon$ that t' is very close to t_i . Although this is a reasonable assumption given the well-behaved nature of the integrated intensity function, the problem of not knowing how close t' is to t_i may tend to reduce one's confidence in the algorithm.

3. Initialization

The initialization time required for the programs implementing the two algorithms has not yet been mentioned. Computation time comparisons did not include compilation or linkage times required for the two algorithms. These initialization times were significantly different, being approximately 16 seconds for Algorithm B and only 8 seconds for Algorithm A. Should simulation of only one or two event streams be required it is likely that Algorithm A may actually require less total time than Algorithm B. The difference between the initialization times could probably be reduced if the programs were to be rewritten in Assembler language.

D. CONCLUSION AND RECOMMENDATIONS

1. Conclusion

For general usage, the Poisson-decomposition and gap statistic method of LEWIS and SHEDLER [Ref. 11] is clearly superior to the time-scale transformation method in generating a non-homogeneous Poisson process with a

degree-two exponential polynomial intensity function.

The main advantages of the Poisson-decomposition and gap statistic algorithm are its speed and its qualification as an exact method of variate generation.

The time-scale transformation method enjoys some advantage in core storage requirements and in lower program initialization time. These advantages are not sufficient to overcome the relative deficiencies of much greater computation time and uncertainty about the accuracy of the approximating mechanism for determining event times.

2. Recommendations

The full potential of the Poisson-decomposition and gap statistic algorithm can not be realized until it includes a better method for selecting an upper limit value c^* for Cases IV, V and VI intensity functions. Algorithms for choosing a c^* value that will be close to the maximum value for the function $\lambda^*(t)$ should be developed and incorporated into Algorithm B. A single variable search technique (such as a Golden Section search or Bisection search) for finding the maximum value of a unimodal function may be appropriate.

The computer program for Algorithm B should be rewritten in Assembler language in order to reduce program initialization time. The program could then be developed into a library routine for general use.

The question of fidelity of the simulated non-homogeneous Poisson process to the true theoretical process should be investigated further. Of special interest is the question of how well the algorithm simulates sparse

event streams. Methods for conducting the dispersion test and the hypothesis test for the mean at very high levels of significance (i.e. $\alpha = .01, .005$) for intervals with a low mean number of events are needed.

APPENDIX A

PROOF OF THE VALIDITY OF SCALING THE INVERSE DISTRIBUTION FUNCTION

Proposition: Let T be a continuous random variable with distribution function $F_T(\cdot)$, such that

$$F_T(t) = 0$$

$$F_T(t) = \frac{\Lambda(t) - \Lambda(0)}{\mu_0} \quad 0 < t \leq t_0$$

$$F_T(t) = 1 \quad t > t_0$$

Let $Y = \mu_0 F_T(T)$ such that $T \in (0, t_0)$ and is a real number. Then the random variable Y is uniformly distributed on the interval $(0, \mu_0)$.

Proof: $F_T(\cdot)$ maps the line segment $[0, t_0]$ onto the interval $[0, 1]$. If $F_T(\cdot)$ is strictly increasing on $(0, t_0)$ then $F_T^{-1}(\cdot)$ exists on the interval $(0, 1)$ and maps $(0, 1)$ onto $(0, t_0)$. Now,

$$\begin{aligned} F_Y(y) &= P[Y \leq y] = P[\mu_0 F_T(T) \leq y] \\ &= P[F_T(T) \leq \frac{y}{\mu_0}] \end{aligned}$$

$$= P[T \leq F_T^{-1}(\frac{y}{\mu_0})]$$

$$= F_T\{F_T^{-1}(\frac{y}{\mu_0})\}$$

$$F_Y(y) = \frac{y}{\mu_0} \quad 0 < y < \mu_0$$

Therefore,

$$\frac{dF_Y(y)}{dy} = f_Y(y) = \frac{1}{\mu_0} \quad 0 < y < \mu_0$$

and Y is uniformly distributed on the interval $(0, \mu_0)$.

APPENDIX B

ERROR FUNCTION AND DAWSON'S INTEGRAL REPRESENTATION OF THE INTEGRATED INTENSITY FUNCTION $\Lambda(t)$

The standard form for the error function is:

$$\frac{2}{\sqrt{\pi}} \int_0^t e^{-u^2} du .$$

Dawson's integral is usually represented as:

$$t^{-2} \int_0^t e^{u^2} du .$$

Both of these integrals may be calculated to any desired degree of accuracy by using the exponential series expansion,

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Thus, e^{u^2} and e^{-u^2} may be written as

$$e^{u^2} = \sum_{n=0}^{\infty} \frac{(u^2)^n}{n!} = \sum_{n=0}^{\infty} \frac{u^{2n}}{n!}$$

and

$$e^{-u^2} = \sum_{n=0}^{\infty} \frac{(-u^2)^n}{n!} = \sum_{n=0}^{\infty} \frac{(-1)^n (u^{2n})}{n!}$$

respectively. Integrating e^{u^2} yields

$$\begin{aligned} \int_0^t e^{u^2} du &= \int_0^t \sum_{n=0}^{\infty} \frac{u^{2n}}{n!} du \\ &= \sum_{n=0}^{\infty} \int_0^t \frac{u^{2n}}{n!} du \\ &= \sum_{n=0}^{\infty} \frac{t^{2n+1}}{(2n+1)n!} \end{aligned}$$

Multiplying by t^{-2} results in

$$t^{-2} \int_0^t e^{u^2} du = \sum_{n=0}^{\infty} \frac{t^{2n-1}}{(2n+1)n!}$$

and the series representation for Dawson's integral is revealed. This series will converge for all t , although the value of the integral increases very rapidly as t increases.

Using the same argument it is easily shown that the series representation for $\int_0^t e^{-u^2} du$ is

$$\sum_{n=0}^{\infty} \frac{(-1)^n t^{2n+1}}{(2n+1)n!} .$$

The error function is obtained by multiplying the integral by the constant $2/\sqrt{\pi}$,i.e.,

$$\frac{2}{\sqrt{\pi}} \int_0^t e^{-u^2} du = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n t^{2n+1}}{(2n+1)n!} .$$

Although the series expansion may be used to calculate both functions, there are very efficient computer library routines available for computing Dawson's integral and the error function.

The FORTRAN supplied procedures ERF and DERF [Ref. 15] evaluate the above function for input values of t.

The IMSL (International Mathematical and Statistical Libraries, Inc.) FUNCTION MMDAW [Ref. 6] evaluates Dawson's integral for input values of t.

It remains to be shown that the intensity function $\lambda(t) = \exp\{\alpha_1 + \alpha_1 t + \alpha_2^2 t\}$ may be integrated over the interval $(0, t_0]$ using either ERF (or DERF) or MMDAW.

Consider,

$$\Lambda(t_0) - \Lambda(0) = \int_0^{t_0} \lambda(t) dt = \int_0^{t_0} \exp(\alpha_0 + \alpha_1 t + \alpha_2 t^2) dt$$

By completing the square of the exponent the expression becomes

$$\Lambda(t_0) = \int_0^{t_0} \exp \left\{ \alpha_0 - \frac{\alpha_1^2}{4\alpha_2} \right\} \exp \left\{ \alpha_2 \left(t + \frac{\alpha_1}{2\alpha_2} \right) \right\} dt$$

($\Lambda(0) = 0$, since 0 is the left end point of the interval over which $\lambda(t)$ is defined).

For $\alpha_2 > 0$,

$$\Lambda(t_0) = \exp \left\{ \alpha_0 - \frac{\alpha_1^2}{4\alpha_2} \right\} \int_0^{t_0} \exp \left\{ \sqrt{\alpha_2} \left(t + \frac{\alpha_1}{2\alpha_2} \right) \right\}^2 dt$$

Letting,

$$u = \sqrt{\alpha_2} \left(t + \frac{\alpha_1}{2\alpha_2} \right)$$

$$du = \sqrt{\alpha_2} dt$$

$$K' = \exp \left\{ \alpha_0 - \frac{\alpha_1^2}{4\alpha_2} \right\}$$

and adjusting the limits of integration, the expression becomes:

$$\Lambda(t_0) = \frac{K'}{\sqrt{\alpha_2}} \int_a^b e^{u^2} du = \frac{K'}{\sqrt{\alpha_2}} \left\{ \int_0^b e^{u^2} du - \int_0^a e^{u^2} du \right\}$$

where,

$$a = (\sqrt{\alpha_2}) \left\{ \frac{\alpha_1}{2\alpha_2} \right\}$$

and

$$b = (\sqrt{\alpha_2}) \left\{ t_0 + \frac{\alpha_1}{2\alpha_2} \right\} .$$

The expression above can be rewritten as follows:

$$\Lambda(t_0) = \frac{K}{\sqrt{\alpha_2}} \left\{ b^2 b^{-2} \int_0^b e^{u^2} du - a^2 a^{-2} \int_0^a e^{u^2} du \right\} .$$

Inside the brackets are two Dawson integral expressions multiplied by constants. Outside the brackets is just one more easily determined constant. Using MMDAW twice and multiplying by the appropriate constants will give the desired integral value.

For $\alpha_2 < 0$,

$$\begin{aligned} \Lambda(t_0) &= \exp \left\{ \alpha_0 - \frac{\alpha_1^2}{4\alpha_2} \right\} \int_0^{t_0} \exp \left\{ -|\alpha_2| \left(t + \frac{\alpha_1}{2\alpha_2} \right)^2 \right\} dt \\ &= \exp \left\{ \alpha_0 - \frac{\alpha_1^2}{4\alpha_2} \right\} \int_0^{t_0} \exp \left\{ -[\sqrt{|\alpha_2|} \left(t + \frac{\alpha_1}{2\alpha_2} \right)]^2 \right\} dt \end{aligned}$$

Substituting as before gives

$$\begin{aligned} \Lambda(t_0) &= \frac{K}{\sqrt{|\alpha|}} \int_a^b e^{-u^2} du \\ &= \frac{K}{\sqrt{|\alpha|}} \frac{\sqrt{\pi}}{2} \left\{ \frac{2}{\sqrt{\pi}} \int_a^b e^{-u^2} du - \frac{2}{\sqrt{\pi}} \int_0^a e^{-u^2} du \right\} \end{aligned}$$

and the error function can be recognized within the brackets.

Two error function calculations, a subtraction and a multiplication by a constant are sufficient to determine $\Lambda(t_0)$.

APPENDIX C

COMPARISON OF GAP STATISTICS ALGORITHM AND TIME-SCALE TRANSFORMATION ALGORITHM FOR SIMULATION OF NON-HOMOGENEOUS POISSON PROCESSES WITH LOG-LINEAR INTENSITY FUNCTIONS

LEWIS and SHEDLER [Ref. 13] present two algorithms for simulation of non-homogeneous Poisson processes with intensity function $\lambda(t) = \exp(\beta_0 + \beta_1 t)$. The first algorithm, which uses a time-scale transformation is easy to employ since the inverse of the integrated intensity function is known explicitly. The second algorithm uses gap statistics to generate event streams.

Both methods are exact. That is, other than the physical precision constraints inherent to the computer, no approximations are necessary in generating event times. However the gap statistics algorithm obviates the need for ordering an array of variates. Also, the gap statistics algorithm takes advantage of a fast standard exponential variate generator, (Random Number Generator Package LLRANDOM, Ref. 12), thereby avoiding the time consuming computation of taking logarithms. The time-scale algorithm must perform both the ordering of variates and the computation of logarithms.

Both algorithms were programmed and tested for computational speed efficiency. The gap statistics algorithm was roughly twice as fast as the time-scale transformation algorithm.

The SUBROUTINE NHPP2 uses the gap statistics algorithm within the Poisson-decomposition and gap statistic algorithm (Algorithm B). It is the presence of SUBROUTINE NHPP2 which markedly reduces the ordering requirements of Algorithm B from those necessary in Algorithm A.

APPENDIX D

POISSON VARIATE GENERATION

A. BACKGROUND

Acknowledgement: The content of this appendix is a paraphrase of portions of Chapter 11 of an unpublished book by J. H. AHRENS and U. DIETER [Ref. 1].

The Poisson distribution has the probability mass function

$$p_i = \frac{e^{-\lambda} \lambda^i}{i!}$$

where p_i is the probability that exactly i events are observed to occur in a unit time interval, given that events arrive at an average rate λ .

The intervals between events in a Poisson process of rate λ are independent and have an exponential distribution with mean $1/\lambda$, i.e.

$$f(x) = \lambda e^{-\lambda x}.$$

The probability integral transform method can be used to obtain a sample of exponential variates from a sample of uniform $(0, 1)$ variates, u_1, u_2, \dots , since

$$x_1 = -\frac{1}{\lambda} \ln u_1, \quad x_2 = -\frac{1}{\lambda} \ln u_2, \quad \dots$$

A λ rate event stream can then be simulated from the exponential variates using the following interpretation:

1st event occurs at x_1

2nd event occurs at $x_1 + x_2$

etc.

This property yields two simple methods for generating a realization on the Poisson random variable. These are the additive method and the multiplicative method.

B. THE ADDITIVE METHOD FOR GENERATING A POISSON VARIATE

The number k of units arriving in a unit interval in a Poisson process with rate λ must be found. This will be the k for which

$$x_1 + x_2 + \dots + x_k \leq 1$$

and

$$x_1 + x_2 + \dots + x_k + x_{k+1} > 1$$

or equivalently, for which

$$-\ln u_1 - \ln u_2 - \dots - \ln u_k \leq \lambda$$

and

$$-\ln u_1 - \ln u_2 - \dots - \ln u_k - \ln u_{k+1} \geq \lambda \quad (1)$$

(Note: Since $u_i \leq 1 \forall i$, $-\ln u_i \geq 0$)

Thus by using uniform variates, computing logarithms, adding and counting, the Poisson variate is obtained. The

AD-A047 164

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 9/2

A COMPARISON OF TWO ALGORITHMS FOR THE SIMULATION OF NON-HOMOGE--ETC(U)

SEP 77 M L PATROW

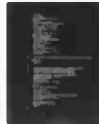
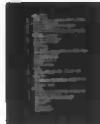
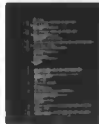
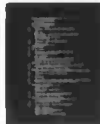
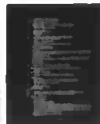
UNCLASSIFIED

NL

2 of 2

AD
A047164

EL



END
DATE
FILMED

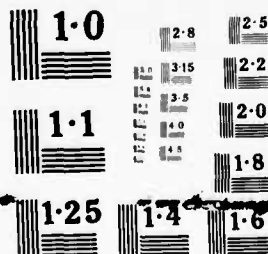
1- 78

DDC

2 OF 2

AD

A047164



problem with the additive method is that if λ is very large, say 2000, it requires considerable effort to generate a single Poisson variate. Although addition is extremely fast on a computer, the computing of logarithms is relatively slow.

C. THE MULTIPLICATIVE METHOD FOR GENERATING A POISSON VARIATE

Multiplying (1) by -1 gives:

$$\ln(u_1 \cdot u_2 \cdot \dots \cdot u_k) \geq -\lambda$$

and

$$\ln(u_1 \cdot u_2 \cdot \dots \cdot u_k \cdot u_{k+1}) < -\lambda ;$$

or equivalently

$$u_1 \cdot u_2 \cdot \dots \cdot u_k \geq e^{-\lambda}$$

and

$$u_1 \cdot u_2 \cdot \dots \cdot u_k \cdot u_{k+1} < e^{-\lambda} .$$

So by generating uniforms, successive multiplication and counting, the Poisson variate k is produced. The multiplicative method eliminates the need for computing logarithms. However trouble is encountered for large values since $e^{-\lambda}$ becomes very small as λ increases. Underflow problems occur for λ values of approximately 175 and larger on the IBM 360/67 computer system using single precision word lengths. The precision problem may be overcome by partitioning the Poisson random variable into the sum of two or more Poisson random variables. However the average number of multiplications and uniform variates required is still proportional to λ . More efficient

methods exist such as the Gamma method.

D. THE GAMMA METHOD FOR GENERATING A POISSON VARIATE

(The following discussion is a condensed account of Ahrens and Dieter's presentation; Ref. 1, pp.11-20,21,22)

In order to obtain a sample k from the Poisson distribution of mean μ , select a positive integer n (typically n is a little smaller than μ). Then take a sample x from the Gamma distribution of parameter n .

(1) If $x > \mu$ return a sample k from the binomial distribution with parameters $n-1, \mu/x$

(2) If $x \leq \mu$ take a sample j from the Poisson distribution of mean $\mu-x$ and return $k \leftarrow n + j$.

The sample x simulates the n -th event (arrival) in a Poisson process of rate 1. If $x > \mu$ then there are $n-1$ arrivals in the interval $(0,x)$, and each of these has a probability of μ/x of being below μ [Case (1)]. If $x \leq \mu$ then the n simulated arrivals are all before μ , and the sample j indicates the additional events between x and μ [Case (2)].

(At this point Ref. 1 includes a formal proof of the procedure, which will be omitted.)

The explicit algorithm relies on an efficient method for sampling from the Gamma distribution. (Such a method has been implemented as W. R. CHURCH Computer Center library routine GAMA by D. W. Robinson and P. A.W. Lewis and is documented in Reference 10.) The constant n is arbitrarily chosen as $n = [0.875 \mu]$. This avoids the Case (1) almost completely if μ is large and a simple method for sampling from the binomial distribution becomes sufficient: the Bernoulli process is simulated in Steps 8-10 below. The algorithm for sampling from the Poisson distribution with mean $\mu - x$ in Case (2) is the simple multiplicative method explained above and is employed in Steps 3-5. However, if $\mu - x$ is still large (larger than the "cut-off point" c) the entire procedure is iterated with a new auxiliary sample x from a Gamma distribution of mean $n' = [0.875 (\mu - x)]$ etc.

Algorithm - The Gamma Method, Ahrens and Dieter

1. Initialize $k \leftarrow 0$, $w \leftarrow \mu$.
2. If $w \geq c$ go to 6 ($c = 16$ was determined empirically by Ahrens and Dieter to be reasonable).
3. (Start Case (2)). Set $p \leftarrow 1$ and calculate $b \leftarrow e^{-w}$.
4. Generate u and set $p \leftarrow p \cdot u$. If $p < b$ deliver k .

5. Increase $k \leftarrow k+1$ and go to 4.
6. Set $n \leftarrow [dw]$ where $d = 7/8$ (d value was also selected empirically). Take a sample x from the Gamma $(n,1)$ distribution. If $x > w$ go to 8.
7. Set $k = k + n$, $w \leftarrow w-x$ and go to 2.
8. (Start Case (1)). Set $p \leftarrow w/x$.
9. Generate u . If $u \leq p$ increase $k \leftarrow k + 1$.
10. Set $n \leftarrow n-1$. If $n > 1$ go to 9.
11. Deliver k .

Ahrens and Dieter claim that the computation time for the Gamma method grows ultimately like $\alpha + \beta \ln \mu$. Computation time for the additive and multiplicative methods grow like μ . Therefore for the large μ values typically demanded in the simulation of non-homogeneous Poisson processes, the Gamma method is signally superior in terms of computation speed.

APPENDIX E

GRAPHICAL PRESENTATION OF NEWTON-RAPHSON METHOD

A. GENERAL PRINCIPLES

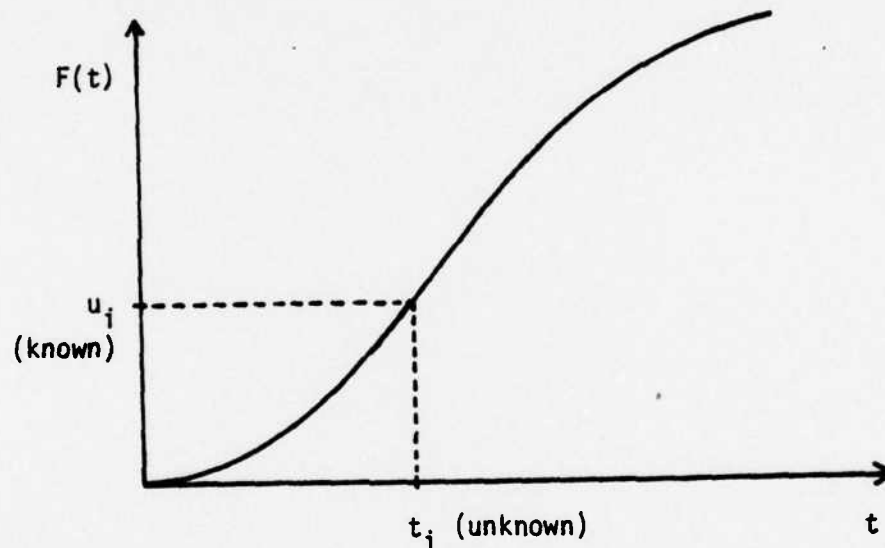


Figure E-1

Consider the function $F(t)$ pictured above in Figure E-1. The objective is to find, for a known value u_i on the vertical axis, a unique corresponding value t_i such that $F(t_i) = u_i$. If an explicit expression for the inverse of $F(\cdot)$ exists, the calculation may be made directly, i.e., $t_i = F^{-1}(u_i)$. Otherwise, numerical methods must be used to approximate t_i .

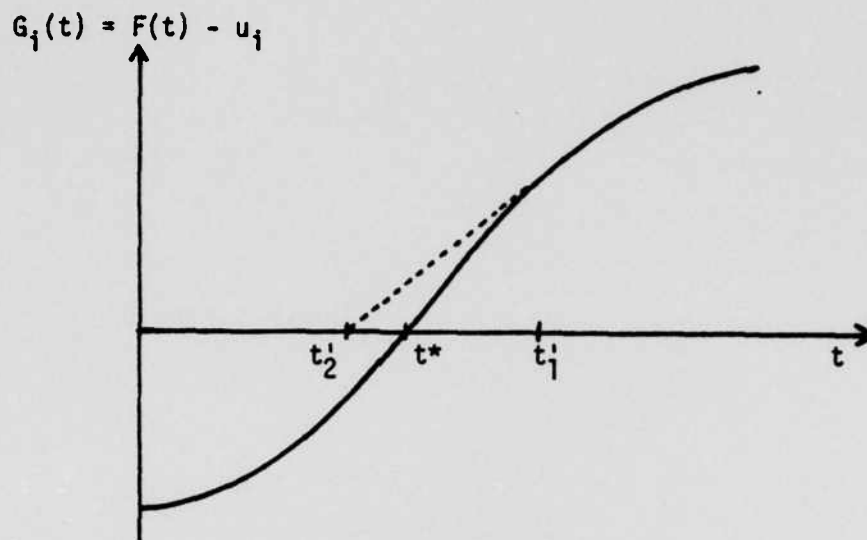


Figure E-2

Form the new function $G_i(t) = F(t) - u_i$ (see Figure E-2). In effect, the horizontal axis has been displaced upward a distance u_i . Now if a t^* can be found such that $G_i(t^*) = 0$, then $F(t^*) = u_i$ and $t^* = t_i$, the desired value.

Assume that an initial approximation for t^* can be made, say t_1' . If $G_i(t_1')$ and $G_i'(t_1') = g_i(t_1')$ can be computed we can write

$$g_i(t_1') = \frac{G_i(t_1')}{(t_1' - t_2')}$$

where t_2' is the intercept of the line tangent to $G_i(\bullet)$ at $G_i(t_1')$ with the t -axis. Solving the above expression for t_2' gives:

$$t_2' = t_1' - \frac{G_i(t_1')}{g_i(t_1')}.$$

It is evident from the graph that t_2' is closer than t_1' to the unknown t^* .

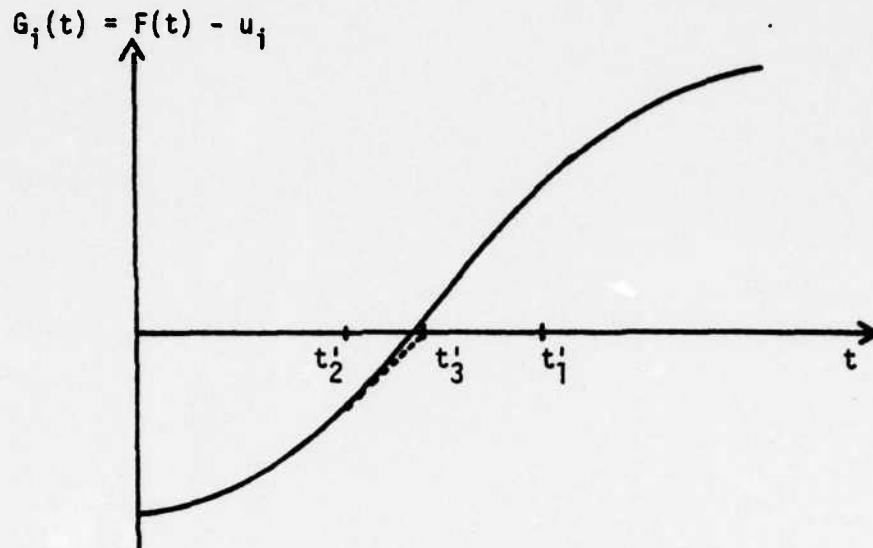


Figure E-3

Repeating the procedure (Figure E-3), using t_2' as a new approximation will yield t_3' a still better approximation of t^* . In general, given an approximation t_k' has been obtained, a closer approximation t_{k+1}' can be calculated by the relationship

$$t_{k+1}' = t_k' - \frac{G_i(t_k')}{g_i(t_k')} .$$

It can be shown that successive approximations will converge to the value t^* provided $G_i(t)$ satisfies certain criteria [Ref. 5, p. 449, 450].

Since $G_i(t)$ has the form of a distribution function $F(t)$ it satisfies all the conditions for convergence except for the case where the interval $[t_1', t_i]$ contains an inflection point. If the inflection point(s) can be identified, this troublesome situation can easily be avoided by handling the function $G_i(\cdot)$ in discrete sections over the discrete intervals, $(0, t_1''), (t_1'', t_2''), \dots, (t_{k-1}'', t_k'')$ where the t_j''

($j = 1, \dots, k$) are values such that $G_i(t_j'')$ are all the inflection points of $G_i(\cdot)$ over $(0, t_0]$. (For the special case under consideration at most one inflection point will be encountered. It is the maximum or minimum of the intensity function.)

Successive approximations are computed until $|G_i(t_k')| < \epsilon$ at which time t_k' is assumed to be close enough to t^* .

B. INITIAL APPROXIMATIONS

The problem of obtaining a good initial approximation for each t_i was solved in the following manner (see Figure E-4 below):

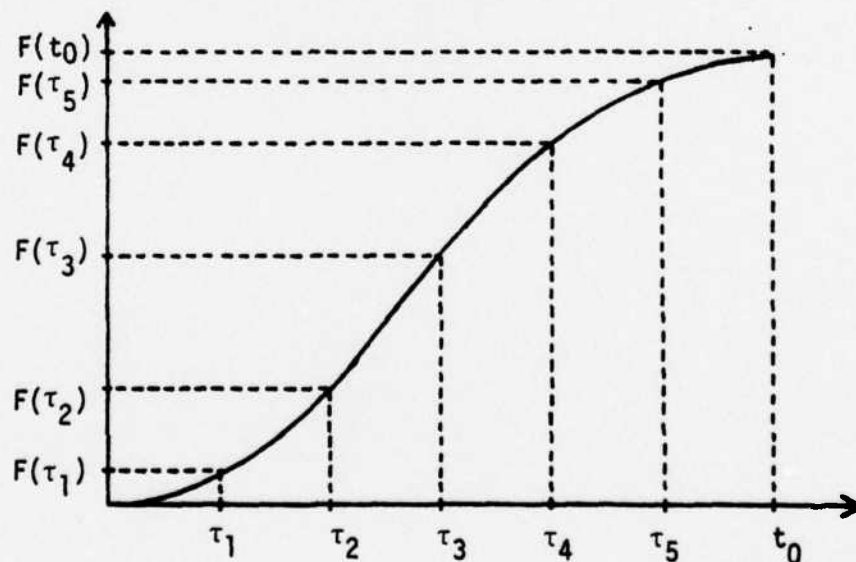


Figure E-4

The interval $(0, t_0]$ was divided into several segments $(0, \tau_1]$, $(\tau_1, \tau_2]$, etc. Then the values $F(\tau_j)$ ($j = 1, 2, \dots$) were computed. Each resulting abscissa and ordinate value was stored in an array, i.e.,

t	F(t)
0	0
τ_1	$F(\tau_1)$
τ_2	$F(\tau_2)$
.	.
.	.
t_0	$F(t_0)$

For each value u the array was searched until two function values were located such that $F(\tau_j) \leq u \leq F(\tau_{j+1})$. Either τ_j or τ_{j+1} was then selected as the initial approximation for the value t_i (i.e. $t_{i,1}$) such that $F(t_i) = u_i$. The function $G_i(t) = F(t) - u_i$ was then formed and the Newton-Raphson iterations performed until the epsilon criterion was satisfied.

.....

SUBROUTINE MTDINV

PURPOSE

SIMULATES A NON-HOMOGENEOUS POISSON PROCESS WITH
QUADRATIC EXPONENTIAL INTENSITY FUNCTION OVER A
GIVEN INTERVAL USING A TIME-SCALE TRANSFORMATION
OF A HOMOGENEOUS POISSON PROCESS.

USAGE

CALL MTDINV (A,A1,A2,EL,ER,IS,II,M,IER)

DESCRIPTION OF PARAMETERS

A - CONSTANT IN INTENSITY FUNCTION.
A1 - 1ST DEGREE COEFF IN INTENSITY FUNCTION.
A2 - 2ND DEGREE COEFF IN INTENSITY FUNCTION.
EL - LEFT END POINT OF INTERVAL.
ER - RIGHT END POINT OF INTERVAL.
IS - RANDOM NUMBER SEED. ANY INTEGER WITH NINE
OR LESS DIGITS.
II - 0 FOR TIMES OF EVENTS.
1 FOR TIMES BETWEEN EVENTS.
M - RETURNED VALUE. EQUALS NUMBER OF EVENTS IN
NON-HOMOGENEOUS POISSON PROCESS.
IER - ERROR CODE. MAY BE WRITTEN ON DEMAND. IER
EQUALS THE NUMBER OF TIMES PROGRAM WAS
FORCED TO ABORT SIMULATION AND START AGAIN
DUE TO EXCESSIVE EVENTS (MORE THAN 5000).

CCMMENTS

CALLING PROGRAM MUST HAVE A COMMON REGION, BLOK1,
OF DIMENSION (5000).

EXAMPLE: DIMENSION T(5000)
COMMON/BLOK1/T

CALLING PROGRAM MUST USE THE FOLLOWING JCL CARDS

// EXEC FORTCLG,IMSL=DP
//FORT.SYSIN DD *

TIMES TO EVENTS OR TIMES BETWEEN EVENTS WILL BE
STORED IN CELLS T(1) THROUGH T(M).

.....

SUBROUTINE MTDINV (A,A1,A2,EL,ER,IS,II,M,IER)
DIMENSION TAU(5000), BRKT(5000,2)
COMMON /BLOK1/ TAU/BLOK2/BRKT
CALL OVFLOW

IER = -1
1 IER = IER+1
BRKT(1,1) = 0.
BRKT(1,2) = 0.

INTEGRATE INTENSITY FUNCTION

CALL HELP (A,A1,A2,EL,ER,PAR)
PAR1 = PAR

GENERATE POISSON VARIATE M

CALL PVAR (IS,PAR1,M)
IF (M.GT.5000) GO TO 1

SET EPSILON VALUE FOR NEWTON-RAPHSON STOPPING RULE

EPS = FLOAT(M+1)*0.00001

```

C      GENERATE M UNIFORM VARIATES; ORDER AND SCALE
C
C      CALL RANDOM (IS,TAU,M)
C      CALL PXSORT (TAU,1,M)
C
C      DO 2 I=1,M
C      TAU(I) = TAU(I)*PAR
2 CCNTINUE
C
C      DETERMINE INTERVAL PARTIONING SCHEME
C      FOR SUBROUTINE BNCHMK
C
C      I2 = M/4
C      DIV = AMAX0(10,I2)
C      IDIV = INT(DIV+2)
C      DELTA = (ER-EL)/DIV
C      ENDL = EL
C
C      CCMPUTE ARRAY OF ABSCISSAS AND ORDINATES FOR
C      INTEGRATED INTENSITY FUNCTION
C
C      CALL BNCHMK (A,A1,A2,ENDL,DELTA,IDIV)
C
C      BEGIN NEWTON-RAPHSON ITERATIONS
C
C      K = 0
C      J = 1
3 K = K+1
4 IF (TAU(K).LT.BRKT(J,2)) GO TO 5
C      J = J+1
C      GO TO 4
5 IF (ABS(BRKT(J,2)-TAU(K)).LT.ABS(BRKT(J-1,2)-TAU(K)))
C      *GO TO 6
C      T = BRKT(J-1,1)
C      ACD = (BRKT(J-1,2)-TAU(K))/EVAL(A,A1,A2,T)
C      GO TO 7
6 T = BRKT(J,1)
C      ADD = (BRKT(J,2)-TAU(K))/EVAL(A,A1,A2,T)
7 IF (ABS(ADD).LT.EPS) GO TO 8
C      T = T-ADD
C      CALL HELP (A,A1,A2,EL,T,X)
C      ADD = (X-TAU(K))/EVAL(A,A1,A2,T)
C      GO TO 7
8 TAU(K) = T
C      IF (K.EQ.M) GO TO 9
C      GO TO 3
9 IF (II.EQ.0) RETURN
C
C      TIMES BETWEEN EVENTS ARE REQUESTED-CALCULATE THEM
C
C      S = TAU(1)
C      TAU(1) = TAU(1)-EL
C
C      DO 10 I=2,M
C      S1 = TAU(I)
C      TAU(I) = TAU(I)-S
C      S = S1
10 CONTINUE
C
C      RETURN
C      END
C
C      SUBROUTINE HELP CALCULATES THE VALUE OF THE
C      INTEGRATED INTENSITY FUNCTION OVER ANY INTERVAL (0,T)
C      OVER WHICH THE NHPP OCCURS
C
C      SLBROUTINE HELP (A,A1,A2,EL,ER,XX)
C      DOUBLE PRECISION MMDAW,BB,AA
C      Z = SQRT(ABS(A2))
C      Y = (A1*Z)/(2.*A2)

```

```

      AA = Y
      CC = A-A1*A1/(4.*A2)
      CC = EXP(CC)/Z
      IF (A2.LT.0.) GO TO 2
      Q1 = DEXP(AA**2)*MMDAW(AA)
C
      DO 1 I=2, IDIV
      BINC = BINC+DELTA
      BB = Z*BINC+Y
      Q2 = DEXP(BB**2)*MMDAW(BB)
      XX = CC*(Q2-Q1)
      BRKT(I,1) = BINC
      BRKT(I,2) = XX
      1 CONTINUE
C
      GO TO 3
      2 CC = CC*.8862269
C
      DO 3 I=2, IDIV
      BINC = BINC+DELTA
      BB = Z*BINC+Y
      XX = CC*(DERF(BB)-DERF(AA))
      BRKT(I,1) = BINC
      BRKT(I,2) = XX
      3 CONTINUE
C
      RETURN
      END
C
C
C
C
C
      FUNCTION EVAL COMPUTES THE VALUE OF THE
      INTENSITY FUNCTION
      FUNCTION EVAL (A,A1,A2,T)
      EVAL = A+A1*T+A2*T**2
      EVAL = EXP(EVAL)
      RETURN
      END

```

.....

SUBROUTINE DEGTWO

PURPOSE

SIMULATES A NON-HOMOGENEOUS POISSON PROCESS WITH
QUADRATIC EXPONENTIAL INTENSITY FUNCTION OVER A
GIVEN INTERVAL USING THE POISSON-DECOMPOSITION
AND GAP STATISTIC ALGORITHM.

USAGE

CALL DEGTWO (IS,A,A1,A2,EL,ER,II,N,IER)

DESCRIPTION OF PARAMETERS

IS - RANDOM NUMBER SEED. ANY INTEGER WITH NINE
OR LESS DIGITS.
A - CONSTANT IN INTENSITY FUNCTION.
A1 - 1ST DEGREE COEFF IN INTENSITY FUNCTION.
A2 - 2ND DEGREE COEFF IN INTENSITY FUNCTION.
EL - LEFT END POINT OF INTERVAL.
ER - RIGHT END POINT OF INTERVAL.
II - 0 FOR TIMES OF EVENTS.
1 FOR TIMES BETWEEN EVENTS.
N - A VECTOR OF LENGTH 5. N(1) THROUGH N(4)
CONTAIN NUMBERS OF EVENTS FROM VARIOUS
COMPONENTS OF THE DECOMPOSED INTENSITY
FUNCTION. N(5) CONTAINS THE TOTAL NUMBER
OF EVENTS IN THE NON-HOMOGENEOUS POISSON
PROCESS.

COMMENTS

CALLING PROGRAM MUST HAVE A COMMON REGION, HOLD,
OF DIMENSION (5000), AND AN INTEGER ARRAY OF
DIMENSION (5).

EXAMPLE: DIMENSION T(5000), N(5)
COMMON/HOLD/T

CALLING PROGRAM MUST CONTAIN THE FOLLOWING
ASSIGNMENT STATEMENT:

M=N(5)

CALLING PROGRAM MUST USE THE FOLLOWING JCL CARDS

// EXEC FORTCLG,IMSL=DP
//FORT.SYSIN DD *

TIMES TO EVENTS OR TIMES BETWEEN EVENTS WILL BE
STORED IN CELLS T(1) THROUGH T(M).

.....

SUBROUTINE DEGTWO (IS,A,A1,A2,EL,ER,II,N,IER)
DIMENSION TIMES(5000), T(5000), N(5), P(5)
COMMON /MIKE/ TIMES/HOLD/T
CALL OVFLOW

INITIALIZE VARIABLES

P(1) = A
P(2) = A1
P(3) = A2
P(4) = 0.
P(5) = 0.

DO 1 I=1,5
1 N(I) = 0

IF RATE FUNCTION IS LESS THAN DEGREE TWO,

```

C      USE NHPP2 ROUTINE ONLY
C      IF (A2.EQ.0.) GO TO 2
      GO TO 4
2     CALL NHPP2 (IS,EL,ER,A,A1,II,N1,IER)
      N(5) = N1
      IF (N1.EQ.0) RETURN
C
      DO 3 I=1,N1
      TIMES(I) = T(I)
3     CONTINUE
      RETURN
C
C      DETERMINE COEFFICIENTS FOR MODIFIED
C      DEGREE ONE RATE FUNCTION
C
4     TEST = -A1/(2.*A2)
      TINT = ER-EL
      IF (A1.GE.0..AND.A2.GT.0.) GO TO 5
      GO TO 6
5     B = A-A2*TINT**2
      B1 = A1+2.*A2*TINT
      GO TO 10
6     B = A
      IF ((A1.LE.0..AND.A2.LT.0.).OR.(A1.GT.0..AND.A2.LT.0.
      *.AND.TEST.GE.TINT)) GO TO 7
      GO TO 8
7     B1 = A1+A2*TINT
      GO TO 10
8     IF (A1.GT.0..AND.A2.LT.0..AND.TEST.LT.TINT) GO TO 9
      B1 = A1
      GO TO 10
9     B1 = A1/2.
C
C      MUST THE INTERVAL BE PARTITIONED?
C
10    IF (A1*A2.LT.0..AND.TEST.LT.TINT) GO TO 11
      ERNEW = ER
      GO TO 12
11    ERNEW = TEST+EL
C
C      GENERATE DEGREE ONE NHPP ON INTERVAL
C
12    BB = B
      BB1 = B1
      CALL NHPP2 (IS,EL,ERNEW,BB,BB1,0,N1,IER)
      N(1) = N1
      IF (N(1).EQ.0) GO TO 14
C
      DO 13 I=1,N1
      TIMES(I) = T(I)
13    CONTINUE
C
C      COMPUTE LENGTH OF INTERVAL AND DETERMINE VALUE
C      OF CSTAR FOR USE IN REJECTION ROUTINE
C
14    Q = ERNEW-EL
      E1 = A
      E2 = A2*Q**2
      E3 = A1*Q
      E4 = A1**2/(4.*A2)
      E5 = A1**2/(2.*A2)
      IF (A1.GE.0..AND.A2.GT.0.) GO TO 15
      IF (A1.LT.0..AND.A2.GT.0..AND.TEST.GE.TINT) GO TO 16
      IF (A1.LT.0..AND.A2.GT.0..AND.TEST.LT.TINT) GO TO 17
      IF (A1.LE.0..AND.A2.LT.0.) GO TO 18
      IF (A1.GT.0..AND.A2.LT.0..AND.TEST.GE.TINT) GO TO 19
      CSTAR = EXP(E1-E4)-EXP(E1)
      GO TO 20
15    CSTAR = EXP(E1)-EXP(E1-E2)
      GO TO 20

```

```

16 CSTAR = EXP(E1+E2+E3)-EXP(E1+E3)
   GO TO 20
17 CSTAR = EXP(E1-E4)-EXP(E1-E5)
   GO TO 20
18 CSTAR = EXP(E1)-EXP(E1+E3+E2)
   GO TO 20
19 CSTAR = EXP(E1+E3+E2)-EXP(E1)

C
C
C   COMPUTE INTEGRAL OF MODIFIED DEGREE TWO RATE FUNCTION
C   OVER INTERVAL
20 CALL HELP (A,A1,A2,EL,ERNEW,PMTR)
   PMTR = PMTR-(EXP(B)*(EXP(B1*ERNEW)-EXP(B1*EL)))/B1

C
C
C   IDENTIFY AS FIRST SUBINTERVAL
NOTE = 1

C
C
C   GENERATE REALIZATION ON POISSON (PMTR) VARIATE
21 CALL PVAR (IS,PMTR,M)
   IF (NOTE.EQ.1) GO TO 22
   GO TO 25

C
C
C   REJECTION ROUTINE USED ON FIRST SUBINTERVAL
22 N(2) = M
   P(4) = B
   P(5) = B1
   IF (N(2).EQ.0) GO TO 24
   CALL REJECT (IS,EL,CSSTAR,P,Q,N(2))

C
C
C   DO 23 I=1,M
   TIMES(N(1)+1) = T(I)
23 CONTINUE

C
C
C   HAS THE INTERVAL BEEN PARTITIONED?
24 IF (ERNEW.EQ.ER) GO TO 34
   GO TO 27

C
C
C   USE REJECTION ROUTINE ON SECOND PART OF INTERVAL
25 N(4) = M
   P(4) = B
   P(5) = B1

C
C
C   IF NO EVENTS OCCURRED BYPASS REJECTION ROUTINE
   IF (N(4).EQ.0) GO TO 35
   Q = ER-ELNEW
   CALL REJECT (IS,ELNEW,CSSTAR,P,Q,N(4))

C
C
C   COPY TIMES OF EVENTS INTO 'TIMES' ARRAY
N4 = N(1)+N(2)+N(3)

C
C
C   DO 26 I=1,M
   TIMES(N4+1) = T(I)
26 CONTINUE

C
C
C   GENERATION OF VARIATES COMPLETE.
   GO TO ORDERING ROUTINE
   GO TO 35

C
C
C   INTERVAL PARTITION WAS REQUIRED. MUST NOW
C   CONSIDER SECOND SUBINTERVAL
   DETERMINE COEFFICIENTS FOR MODIFIED
   DEGREE ONE RATE FUNCTION

```

```

27 IF (A1.GT.0..AND.A2.LT.0.) GO TO 28
   B = A-A2*TINT**2
   B1 = A1+2.*A2*TINT
   GO TO 29
28 B = A+(A1/2.)*TINT
   B1 = A1/2.+A2*TINT
29 ELNEW = ERNEW

C
C   GENERATE DEGREE ONE NHPP ON INTERVAL
C
   BB = B
   BB1 = B1
   CALL NHPP2 (IS,ELNEW,ER,BB,BB1,0,N3,IER)
   N(3) = N3
   IF (N(3).EQ.0) GO TO 31
   N3 = N(1)+N(2)

C
C   TRANSFER TIMES BETWEEN ARRAYS
C
   DO 30 I=1,N3
   TIMES(N3+1) = T(I)
30 CONTINUE

C
31 Q = TINT

C
C   DETERMINE VALUE OF CSTAR FOR USE IN
C   THE REJECTION ROUTINE
C
   E2 = A2*Q**2
   E3 = A1*Q
   IF (A1.GT.0..AND.A2.LT.0.) GO TO 32
   CSTAR = EXP(E1-E4)-EXP(E1-E5-E3-E2)
   GO TO 33
32 CSTAR = EXP(E1-E4)-EXP(E1+E3+E2)

C
C   COMPUTE INTEGRAL OF MODIFIED DEGREE TWO RATE
C   FUNCTION OVER SECOND INTERVAL
C
33 CALL HELP (A,A1,A2,ELNEW,ER,PMTR)
   PMTR = PMTR-(EXP(B)*(EXP(B1*ER)-EXP(B1*ELNEW)))/B1

C
C   IDENTIFY AS SECOND SUBINTERVAL
C
   NOTE = 2
   GO TO 21

C
C   PARTITION OF INTERVAL NOT REQUIRED. COMPUTE TOTAL
C   EVENTS AND SUPERPCSE TWO EVENT STREAMS
C
34 N(5) = N(1)+N(2)
   IF (N(2).EQ.0) GO TO 38
   LBGN = N(1)+1
   JBGN = 1
   CALL COLATE (LBGN,N(5),1)
   GO TO 38

C
C   PARTITION WAS REQUIRED. DETERMINE
C   AMOUNT OF SORTING NEEDED
C
35 N(5) = N(1)+N(2)+N(3)+N(4)
   IF (N(2).EQ.0.AND.N(4).EQ.0) GO TO 38
   IF (N(4).EQ.0) GO TO 36
   IF (N(2).EQ.0) GO TO 37

C
C   MUST SUPERPOSE FOUR EVENT STREAMS
C
   LBGN = N(1)+1
   LFIN = N(1)+N(2)
   CALL COLATE (LBGN,LFIN,1)
   LBGN = LFIN+N(3)+1
   JBGN = LFIN+1
   CALL COLATE (LBGN,N(5),JBGN)

```

```

C      GO TO 38
C      MLST SUPERPOSE FIRST HALF OF ARRAY ONLY
C
36  N2 = N(1)+N(2)
    LBGN = N(1)+1
    CALL COLATE (LBGN,N2,1)
    GO TO 38
C
C      MUST SUPERPOSE SECOND HALF OF ARRAY ONLY
C
37  KK = N(1)+N(2)+1
    LBGN = N(1)+N(2)+N(3)+1
    LFIN = N(5)
    CALL COLATE (LBGN,N(5),KK)
    GO TO 38
C
C      ARE TIMES OF EVENTS OR TIMES BETWEEN EVENTS REQUESTED?
C
38  IF (II.EQ.0) RETURN
C      CALCULATE TIMES BETWEEN EVENTS
C
    S = TIMES(1)
    TIMES(1) = TIMES(1)-EL
    N5 = N(5)
C
    DO 39 I=2,N5
    S1 = TIMES(I)
    TIMES(I) = TIMES(I)-S
    S = S1
39  CONTINUE
    RETURN
    END
C
C      SUBROUTINE NHPP2 SIMULATES A NON-HOMOGENEOUS
C      POISSON PROCESS WITH A LOG-LINEAR INTENSITY
C      (RATE) FUNCTION
C
    SUBROUTINE NHPP2 (IS,EL,ER,A,A1,II,N,IER)
    DIMENSION T(5000)
    COMMON /HOLD/ T
C
    CALL OVFLOW
C
    INITIALIZE VARIABLES
C
    IER = 0
    TINT = ER-EL
    A = EXP(A+A1*EL)
C
    IS THE POISSON PROCESS HOMOGENEOUS?
C
    IF (A1.EQ.0.) GO TO 3
    PAR = (A*(EXP(TINT*A1)-1.))/A1
    IF (A1.GT.0.) GO TO 1
    IFLAG = 3
    GO TO 2
1  A = A*EXP(TINT*A1)
   A1 = -A1
   IFLAG = 2
C
C      CCMPUTE PARAMETERS OF BOTH POISSON RANDOM VARIABLES
C
2  THETA = -A/A1
   GO TO 4
C
C      COMPUTE RATE AND SCALING PARAMETERS FOR HOMOGENEOUS
C      POISSON PROCESS

```

```

3 PAR = TINT*A
  IFLAG = 1
  AINVR = 1./A
C
C
C   COMPUTE NUMBER OF EXPONENTIAL VARIATES REQUIRED
4 NMAX = PAR+6.*SQRT(PAR)
  IS THIS A HOMOGENEOUS POISSON PROCESS?
  IF (IFLAG.EQ.1) GO TO 17
  GENERATE REALIZATION ON POISSON (THETA) VARIATE
5 CONTINUE
  CALL PVAR (IS,THETA,M)
  IF (M.EQ.0) GO TO 7
C
C   CALCULATE TIMES OF EVENTS
  CALL SEXPON (IS,T,NMAX)
  B = -A1
  V = 0.
  JMAX = NMAX+1
C
C   DO 6 I=1,JMAX
C
C   HAVE NUMBER OF EVENTS EXCEEDED THE MAXIMUM NUMBER
C   THAT THE ARRAY CAN HOLD?
  IF (I.GT.NMAX) GO TO 8
  V = V+T(I)/((M-I+1)*B)
  IF (V.GT.TINT) GO TO 9
  T(I) = V
  IF (I.EQ.M) GO TO 10
6 CONTINUE
C
C   NO EVENTS OCCURRED
7 N = 0
  RETURN
C
C   TOO MANY EVENTS FOR ARRAY. INCREMENT ERROR
C   CODE AND TRY AGAIN
8 IER = IER+1
  GO TO 5
C
C   THE NUMBER OF EVENTS OBSERVED TO OCCUR IN THIS
C   NON-HOMOGENEOUS POISSON PROCESS IS 'N'
9 N = I-1
  IF (N.EQ.0) RETURN
  GO TO 11
10 N = M
11 CCNTINUE
C
C   IS THE RATE FUNCTION INCREASING OR DECREASING?
  IF (IFLAG.EQ.3) GO TO 13
  TIME REVERSAL TECHNIQUE IS NECESSARY
  DETERMINE WHETHER N IS EVEN OR ODD
  ISIG = MOD(N,2)
  NLOOP = N/2
  N1 = N+1
C
  DO 12 I=1,NLOOP
    S = T(I)
    T(I) = ER-T(N1-I)
    T(N1-I) = ER-S

```

```

C      12 CONTINUE
C      IF (ISIG.EQ.1) T(NLOOP+1)=ER-T(NLOOP+1)
C      ARE TIMES OF EVENTS REQUESTED?
C      IF (II.EQ.0) RETURN
C      GO TO 15
13     IF (II.NE.0) GO TO 15
C      IF (EL.EQ.0.) RETURN
C      DO 14 I=1,N
C      T(I) = EL+T(I)
14     CCNTINUE
C      RETURN
C      CALCULATE TIMES BETWEEN EVENTS
C      15 S = T(1)
C      DO 16 I=2,N
C      S1 = T(I)
C      T(I) = T(I)-S
C      S = S1
16     CONTINUE
C      RETURN
C      THE POISSON PROCESS IS HOMOGENEOUS
C      17 I = 1
C      U = 0.
C      CALL SEXPON (IS,T,NMAX)
18     U = U+T(I)
C      IF (U.GT.PAR) GO TO 20
C      I = I+1
C      IF (I.GT.NMAX) GO TO 19
C      GO TO 18
C      INCREMENT ERROR CODE
C      19 IER = IER+1
C      TRY AGAIN WITH NEW STRING OF VARIATES
C      GO TO 17
20     N = I-1
C      IF (N.EQ.0) RETURN
C      IF (II.EQ.1) GO TO 22
C      DO 21 I=1,N
C      EL = EL+AINVRS*T(I)
C      T(I) = EL
21     CONTINUE
C      RETURN
C      22 DO 23 I=1,N
C      T(I) = T(I)*AINVRS
23     CONTINUE
C      RETURN
C      END
C      SUBROUTINE PVAR GENERATES A POISSON (THETA)
C      VARIATE, M, USING THE GAMMA METHOD
C      SUBROUTINE PVAR (IS,THETA,M)
C      DIMENSION T(5000)
C      COMMON /HOLD/ T

```

```

      K = 0
      C = 16.
      D = .875
1     IF (THETA.LT.C) GO TO 2
      GO TO 5
2     U = 1.
      CTN = EXP(-THETA)
      MMAX = THETA+6.*SQRT(THETA)
3     I = 1
      CALL SRAND (IS,T,MMAX)
4     U = U*T(I)
      IF (U.LT.CTN) GO TO 8
      I = I+1
      K = K+1
      IF (I.GT.MMAX) GO TO 3
      GO TO 4
5     NP = INT(D*THETA)
      AN = FLOAT(NP)
      CALL GAMA (AN,IS,G,1)
      IF (G.GT.THETA) GO TO 6
      K = K+NP
      THETA = THETA-G
      GO TO 1
6     U = THETA/G
      NP = NP-1
      CALL SRAND (IS,T,NP)
C
      DO 7 I=1,NP
      IF (T(I).LT.U) K = K+1
7     CONTINUE
C
C
C
      THE VALUE M IS ASSUMED BY THE POISSON (THETA) VARIATE
8     M = K
      RETURN
      END
C
C
C
C
C
      SUBROUTINE REJECT GENERATES AN ORDERED SAMPLE
      OF GIVEN SIZE FROM THE SECOND COMPONENT
      OF THE ORIGINAL INTENSITY FUNCTION
      USING A REJECTION-ACCEPTANCE ALGORITHM
C
      SUBROUTINE REJECT (IS,EL,CSTAR,PVEC,Q,L)
      DIMENSION V(500), PVEC(5)
      DIMENSION T(5000)
      COMMON /HOLD/ T
      L20 = L*10
      IF (L20.GT.500) L20=500
      L1 = L+1
      K = 1
1     J = 0
      CALL SRAND (IS,V,L20)
C
      DO 2 I=1,L20
      J = J+1
      T(K) = Q*V(J)+EL
      J = J+1
      IF (V(J).LT.CALC(PVEC,T(K))/CSTAR) K=K+1
      IF (K.EQ.L1) GO TO 3
      IF (J.GE.L20-1) GO TO 1
2     CONTINUE
C
      IF (K.LT.L) GO TO 1
3     CALL PXSORT (T,1,L)
      RETURN
      END
C
C
C

```

```

C      SUBROUTINE HELP EVALUATES THE INTEGRATED INTENSITY
C      FUNCTION OVER THE INTERVAL (EL,ER)

```

```

      SUBROUTINE HELP (A,A1,A2,EL,ER,XX)
      DCUBLE PRECISION MMDAW,BB,AA
      Z = SQRT(ABS(A2))
      Y = (A1*Z)/(2.*A2)
      AA = Z*EL+Y
      BB = Z*ER+Y
      CC = A-A1*A1/(4.*A2)
      CC = EXP(CC)/Z
      IF (A2.LT.0.) GO TO 1
      Q1 = DEXP(AA**2)*MMDAW(AA)
      Q2 = DEXP(BB**2)*MMDAW(BB)
      XX = CC*(Q2-Q1)
      RETURN
1 CC = CC*.8862269
  XX = CC*(DERF(BB)-DERF(AA))
  RETURN
  END

```

```

C
C
C
C
C

```

```

      SUBROUTINE COLATE SUPERPOSES TWO ORDERED
      EVENT STREAMS OVER THE SAME INTERVAL

```

```

      SUBROUTINE COLATE (LBGN,LFIN,JBGN)
      DIMENSION TIMES(5000), T(5000)
      COMMON /MIKE/ TIMES/HOLD/T
      I = JBGN
      J = I
      K = LBGN
1 IF (TIMES(I).LT.TIMES(K)) GO TO 2
  T(J) = TIMES(K)
  J = J+1
  K = K+1
  IF (K.GT.LFIN) GO TO 3
  GO TO 1
2 T(J) = TIMES(I)
  J = J+1
  I = I+1
  IF (I.EQ.LBGN) GO TO 5
  GO TO 1
3 II = LBGN-1
C
  DO 4 N=I,II
    T(J) = TIMES(N)
    J = J+1
4 CONTINUE
  RETURN
5 CONTINUE
  DO 6 N=K,LFIN
    T(N) = TIMES(N)
6 CONTINUE
C
  RETURN
  END

```

```

C

```

```

C
C
C
C
C

```

```

      FUNCTION CALC EVALUATES THE SECOND COMPONENT OF
      THE DECOMPOSED INTENSITY FUNCTION
      FOR ANY INPUT VALUE.

```

```

      FUNCTION CALC (P,ABSA)
      DIMENSION P(5)
      X = P(1)+P(2)*ABSA+P(3)*ABSA**2
      XX = P(4)+P(5)*ABSA
      CALC = EXP(X)-EXP(XX)
      RETURN
      END

```

LIST OF REFERENCES

1. Ahrens, J.H. and Dieter, U., Non-Uniform Random-Numbers, 11-1,2 and 11-20 to 11-23, Institut fur Math. Statistik, Technische Hochschule, Graz, Austria, 1974.
2. Cox, D. R. "The Statistical Analysis of Dependencies in Point Processes." In Stochastic Point Processes, ed. P.A.W. Lewis, 55-66. Wiley, 1972.
3. Cox, D.R. and Lewis, P.A.W., The Statistical Analysis of Series of Events, 17-36, Methuen, London and Wiley, New York, 1966.
4. Fisher R.A., "The Significance of Deviations from Expectations in a Poisson Series", Biometrics, v. 6, p. 17-24, 1950.
5. Hildebrand, F.B., Introduction to Numerical Analysis, 443-450, McGraw-Hill, 1956.
6. IMSL Reference Manual, 6th ed., v. II, (FORTRAN IV) IBM S/370-360, International Mathematical and Statistical Libraries, 1977.
7. Larson, H.J., Introduction to Probability Theory and Statistical Inference, 2d ed., 81-97, Wiley, 1974.
8. Lewis, P.A.W., Introduction to Simulation, eight 80-minute lectures presented at Free University, Amsterdam, December 1975.
9. Lewis F.A.W. "Recent Results in the Statistical Analysis of Univariate Processes." In Stochastic Point

Processes, ed. P.A.W. Lewis, 1-54. Wiley, 1972.

10. Lewis P.A.W. and Robinson D.W., Generating Gamma and Cauchy Random Variables: An Extension to the Naval Postgraduate School Random Number Package, (undated, no reference number) .
11. Lewis, P.A.W., and Shedler, G.S., Simulation of Non-Homogeneous Poisson Process with Degree-Two Exponential Rate Function, IBM Research Division, 1977.
12. Naval Postgraduate School Report NPS55Lw73061A, Naval Postgraduate School Random Number Generator Package LLRANDOM, by G. Learmonth and P.A.W. Lewis, 1973.
13. Naval Postgraduate School Report NPS55Lw75061, Simulation of Non-Homogeneous Poisson Processes with Log-Linear Rate Function, by P.A.W. Lewis and G.S. Shedler, June 1975.
14. Parzen, E., Stochastic Processes, 140, Holden-Day, 1962.
15. Systems, IBM System/360 and System/370 FORTRAN IV Language, GC28-6515-10, 11th ed., p. 121, IBM, 1974.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	2
4. Assoc. Professor A.P. Andrus, Code 55As Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
5. LT D.R. Bouchoux, USN, Code 55Bh Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
6. Assoc. Professor G.H. Bradley, Code 55Bz Department of Operations Research Naval Postgraduate School Monterey, California 93940	1

- | | | |
|-----|-----------------------------------|---|
| 14. | Miss Rosemarie Stampfel, Code 55 | 1 |
| | Department of Operations Research | |
| | Naval Postgraduate School | |
| | Monterey, California 93940 | |
| 15. | Mr. G.S. Shedler | 1 |
| | IBM Research Laboratory | |
| | IBM Building | |
| | San Jose, California 95193 | |
| 16. | Mr. E.N. Ward, Code 0141 | 1 |
| | Manager, Systems Programming | |
| | Computer Center | |
| | Naval Postgraduate School | |
| | Monterey, California 93940 | |
| 17. | Captain Michael L. Patrow, USMC | 2 |
| | 13119 Orleans Street | |
| | Woodbridge, Virginia 22192 | |

ATE
LMED
7